Gradivo in navodila za vaje pri predmetu Seminar iz načrtovanja in razvoja programske opreme v telekomunikacijah

Grega Jakus

15-10-2013

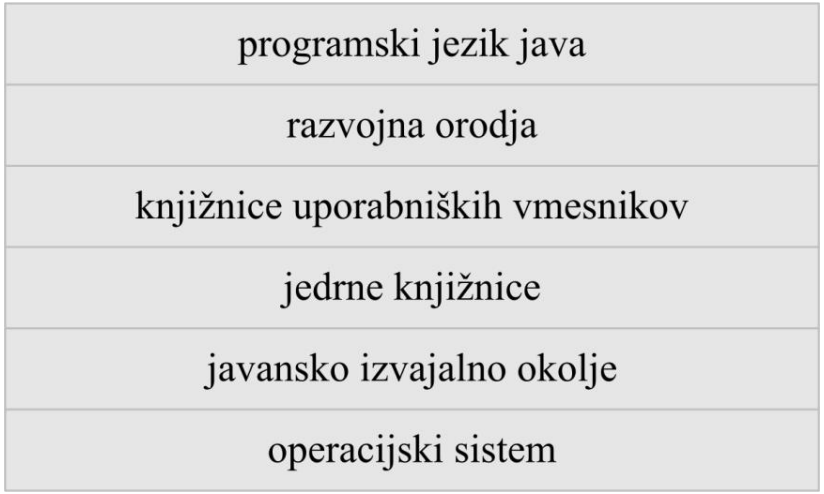
Predstavitev osnovnih pojmov

- Java
- Objektno programiranje
- Programski vmesniki
- Program in aplikacija
- Razvojno okolje Eclipse

- Sun Microsystems, 1995
- Beseda “java” označuje
  - programski jezik
  - programsko platformo
- “programska platforma” ?
  - Programska oprema, ki ponuja osnovno funkcionalnost (npr. dostop do vhodno/izhodnih naprav).
  - Ta funkcionalnost je lahko (selektivno) izkoriščena in nadgrajena z dodatno (uporabniško) programsko kodo z namenom izdelave konkretne aplikacije.

Javanska platforma

- Platforma obsega
  - programski jezik,
  - izvajalno okolje in
  - programske knjižnice
- Javanska platforma je ena izmed najbolj popularnih platform za razvoj spletnih aplikacij, ki temeljijo na arhitekturi odjemalec – strežnik.



- Java ima vsaj 4 okuse:
  - J2SE - standardna različica za osebne računalnike
  - J2ME – različica za mini naprave (mobilni telefoni, pametni televizorji, ...)
  - J2EE – različica za strežniške aplikacije
  - Java Card – Java za pametne kartice in podobne naprave

Programski jezik Java

- Objektno usmerjen visokonivojski programski jezik
- Programska koda, napisana v javi, je prenosljiva med strojnimi platformami in operacijskimi sistemi
- Skladnja podobna skladnji jezika C

```
public class PozdravljenSvet {

   public static void main(String[] args) {

      System.out.println("Pozdravljen svet!");

   }

}
```

Ime datoteke s programsko kodo, se mora ujemati z imenom glavnega razreda

Izvajalno okolje

- Izvorna koda javanske aplikacije se prevede v vmesno kodo (ang. bytecode), ki je prenosljiva med različnimi operacijskimi sistemi in računalniškimi arhitekturami.
- Vmesno kodo neposredno pred izvajanjem javanski navidezni stroj (ang. Java Virtual Machine, JVM) sproti tolmači v izvršni kodi sistema, na katerem se aplikacija izvaja.
- Javansko izvajalno okolje skrbi tudi za nekatere storitve, za katere bi sicer moral poskrbeti razvijalec sam:
  - zagotavljanje varnosti
  - upravljanje s pomnilnikom
  - optimizirano izvajanje kode s pomočjo "pravočasnega" (ang. just in time, JIT) prevajanja namesto klasičnega tolmačenja
- **Izvajalno okolje skupaj z osnovnimi knjižnicami je del javanske distribucije JRE (Java Runtime Environment), ki omogoča poganjanje javanskih aplikacij**
  (ne pa tudi njihovega razvoja)

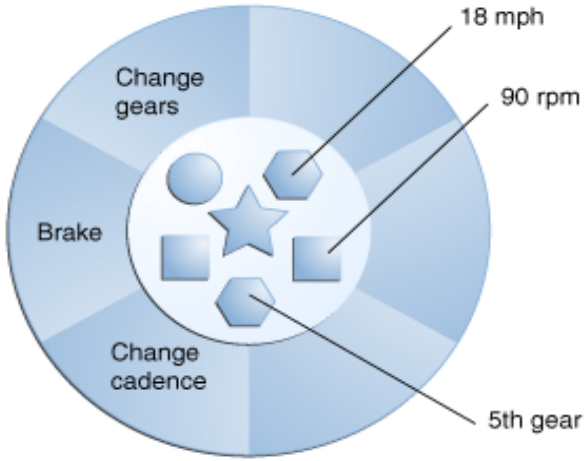
Programske knjižnice

- Z uporabo obsežnega nabora programskih knjižnic lahko dostopamo do že pripravljene obsežne funkcionalnosti, kar pohitri razvoj aplikacij.
- Primeri
  - **`java.lang`** vsebuje razrede in vmesnike, ki predstavljajo osnovo jezika java in so tesno povezani z izvajalnim okoljem
  - **`java.io`** vsebuje razrede, namenjene dostopanju do datotečnega sistema ter različne vhodne in izhodne tokove.
  - **`java.math`** vsebuje matematične funkcije ter razrede za izvajanje matematičnih operacij s poljubno natančnostjo
  - **`java.awt`** in **`java.swing`** vsebujeta sredstva za izdelavo grafičnih uporabniških vmesnikov in 2D grafike
- **Javanske programske knjižnice so na voljo v okviru distribucij Java SDK**
  - **imenovanih tudi JDK – Java Development Kit**
  - **Te poleg programskih knjižnic vsebujejo tudi izvajalno okolje (JRE), zato omogočajo tako izdelavo kot tudi poganjanje javanskih aplikacij**

Objektno programiranje

- Objektno (predmetno) programiranje je vzorec izdelave programov, pri katerem so gradniki programa objekti (predmeti)
- Objekti posnemajo resnične predmete (npr. uporabnika spletne trgovine) ali pa le vsebinsko zaključene dele programa, ki sicer nimajo prispodobe v resničnem svetu (npr. objekt-posrednik pri dostopu do podatkovne zbirke)
- Objekti imajo *stanje* in *obnašanje*
  - stanje je shranjeno v spremenljivkah, ki jim pravimo *lastnosti*
  - obnašanje je mogoče nadzirati preko objektu lastnih funkcij, oziroma *metod*
  - prispodoba s samostalniki in glagoli

  (povečati hitrost, spremeniti prestavo)



- **Objekt je torej skupek podatkov (v obliki lastnosti) in postopkov za njihovo uporabo (v obliki metod)**

Sporočila in vmesniki

- Program je sestavljen iz več objektov, ki medsebojno komunicirajo preko *sporočil*.
- Sporočilo je klic ene izmed objektovih "javnih" metod
  - javne metode so tiste metode, ki jih objekt izpostavlja ostalim objektom.

- Javne metode predpisujejo način interakcije objekta z "zunanjim svetom" oziroma njegov *javni vmesnik* (ang. public interface)
- Vmesnik omogoča, da je nek objekt mogoče vedno uporabiti na enak način
  - (notranje) delovanje objekta se lahko spremeni, a njegova uporaba ostane enaka
- Primer: kolo
  - kot otroci se naučimo voziti otroško kolo, a lahko nato zlahka "presedlamo" na odraslega
- Vmesnik je pogodba med objektom in njegovimi uporabniki
  - "*Objekt se obveže, da bo deloval kot obljublja,*
    če ga bo njegov uporabnik uporabil na predpisan način"
  - Nadzor nad izpolnjevanjem pogodbe opravlja javanski prevajalnik (ang. compiler)

Aplikacijski programski vmesnik (API)

- *Aplikacijski programski vmesnik* je pogodba med programskimi komponentami, ki zagotavlja, da lahko le-te medsebojno komunicirajo
  - Programske komponente navadno vsebujejo več objektov
- API je specifikacija javne vsebine objektov (metod in lastnosti)
- API lahko predstavlja tudi idejo, kako nek objekt uporabiti, kakšen je rezultat uporabe, navadno pa vsebuje bolj tehnološka navodila, npr. katero metodo klicati za dosego zastavljenega cilja.
- Pogovorno kratico "API" uporabimo, ko se sklicujemo na način uporabe in razširitve programskih ogrodij in platform

Program vs. aplikacija

- Program je niz ukazov, ki je sestavljen z namenom izvršitve nekega opravila v okviru računalniškega sistema
- Aplikacija je programska oprema, ki omogoča koristno (produktivno) uporabo računalnika
  - Med aplikacije ne sodi programska oprema, ki skrbi za delovanje računalniškega sistema
  - Med aplikacije tudi ne sodijo virusi in podobni zlonamerni programi
- V grobem je aplikaciji nasprotni pojem *sistemska programska oprema*
  - To je programska oprema, ki povezuje računalniške funkcije, jih upravlja in na ta način omogoča delovanje računalniškega sistema
  - Sistemska programska oprema ne opravlja opravil, ki bi bile neposredno koristne za uporabnika
  - Sistemska programska oprema nudi podporo aplikacijam, te pa nudijo podporo uporabnikom

Razvojno okolje Eclipse

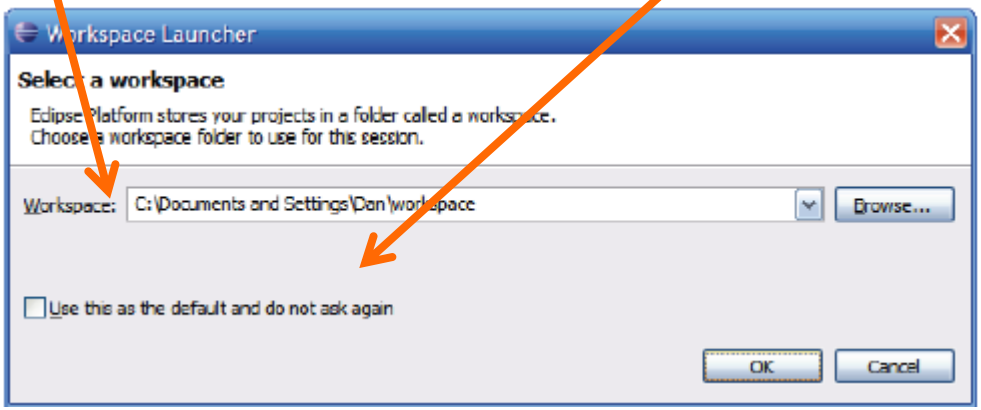
- Odprtokodna platforma, v osnovi namenjena izdelavi t.i. “rich-client” aplikacij
- Osnovne funkcionalnosti platforme razširjamo s pomočjo vtičnikov (ang. plugins)
- Najpogostejša uporaba je kot razvojno okolje za različne programske jezike (java, C, PHP,…)
- V praksi je Eclipse sinonim za razvojno okolje za programski jezik java
- www.eclipse.org

Namestitev okolja Eclipse

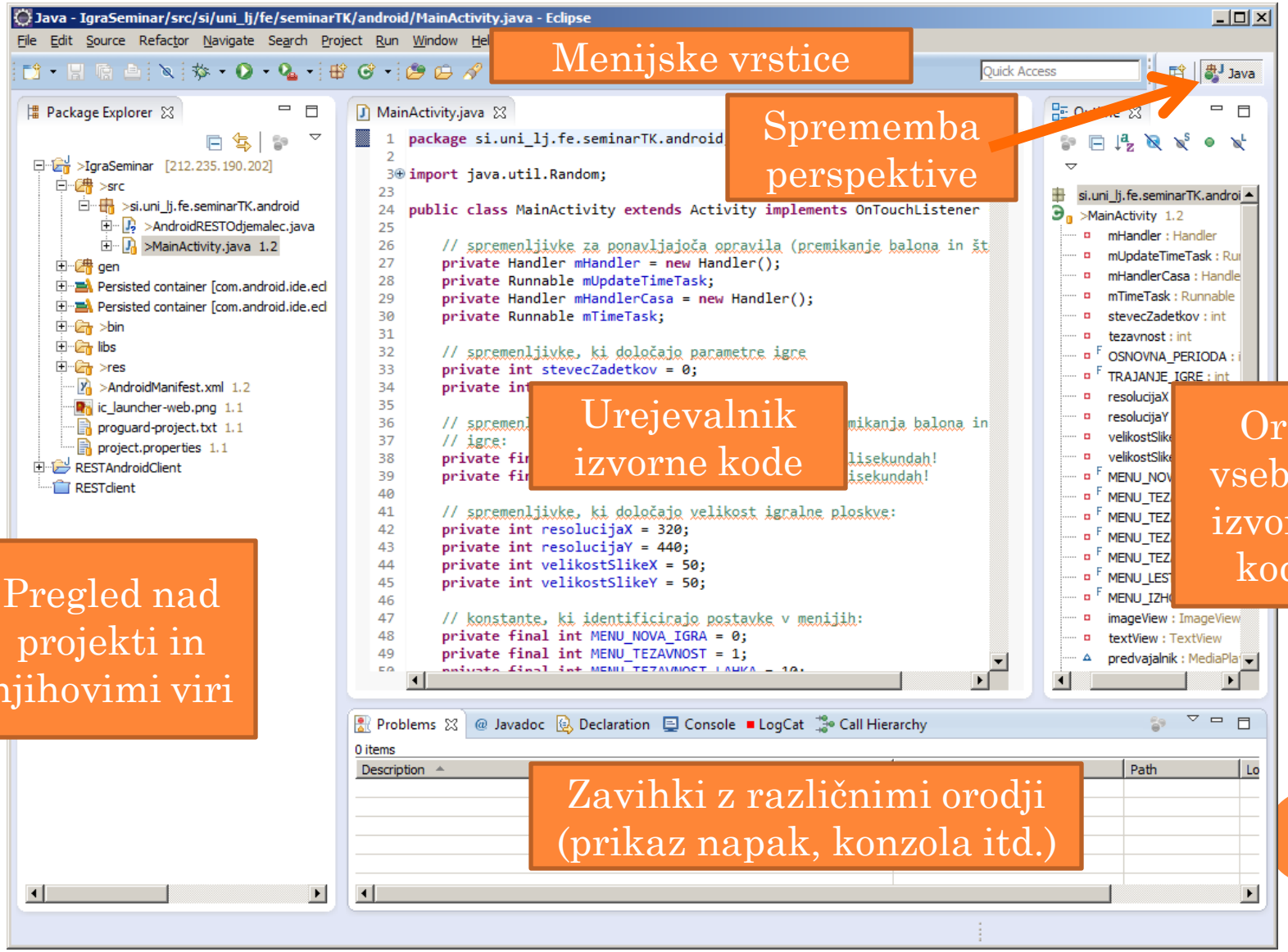
- Eclipse je napisan v javi in zato potrebuje JRE ali SDK
- Obstaja več distribucij okolja Eclipse
  - http://www.eclipse.org/downloads/
  - Namestili bomo distribucijo, namenjeno delu s platformo Java EE: Eclipse IDE for Java EE Developers
  - Zip arhiv razširimo v mapo D:/seminar/eclipse

Delovni prostor (workspace)

- Ko poženete Eclipse, vas ta povpraša po poti do "delovnega prostora"
- Delovni prostor (workspace) je mesto, kamor se bo shranjevala izvorna koda in ostali podatki
- Izberite pot d:\seminar\workspace in nastavite izbrani delovni prostor za privzetega

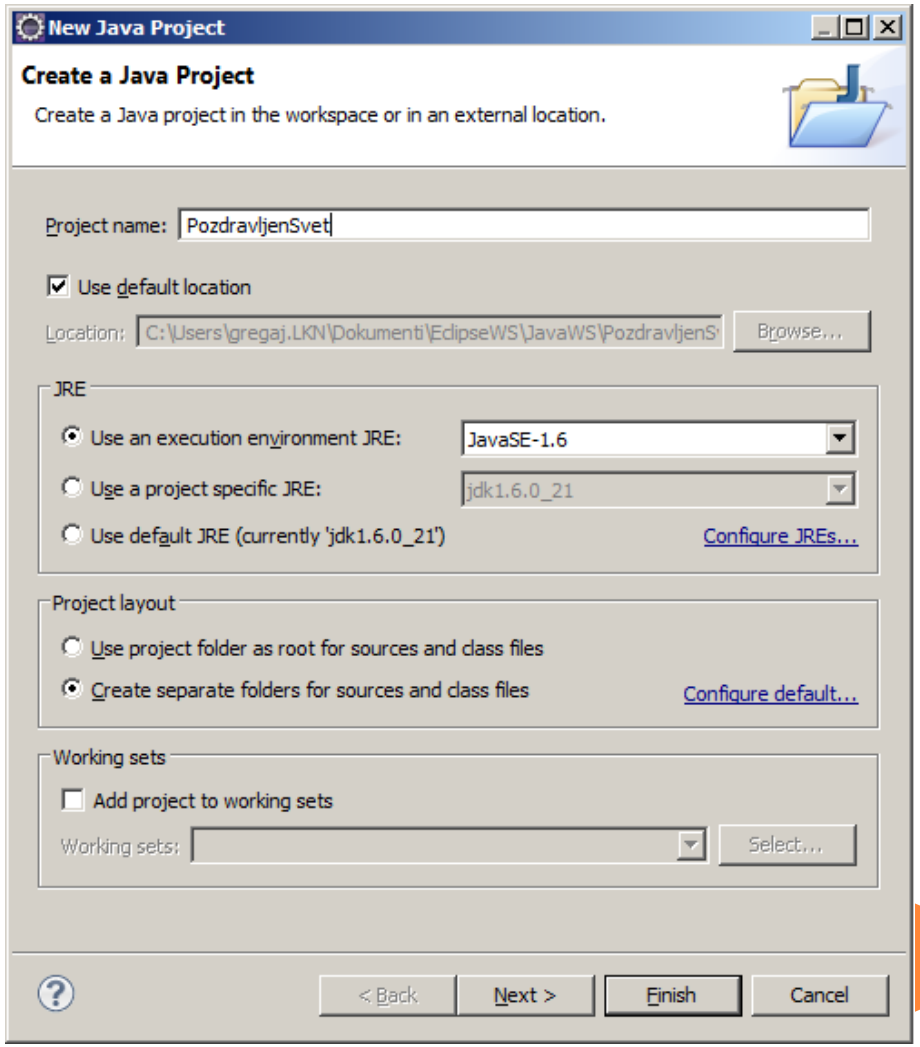


Delovna miza (workbench)



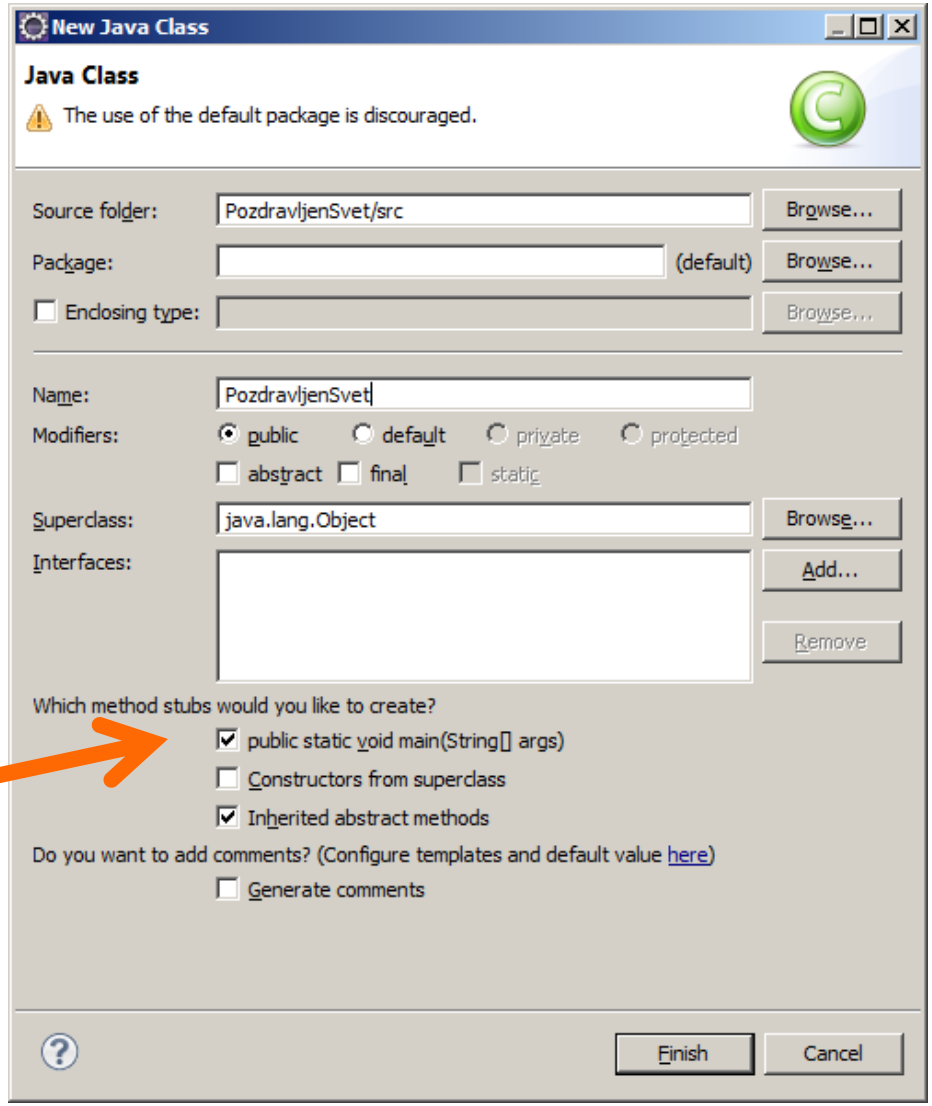
JAVANSKI PROJEKT

- Vsaka aplikacija mora biti vključena v "projekt"
- Ustvarjanje novega projekta:
  File → New → (Other...) → Java Project

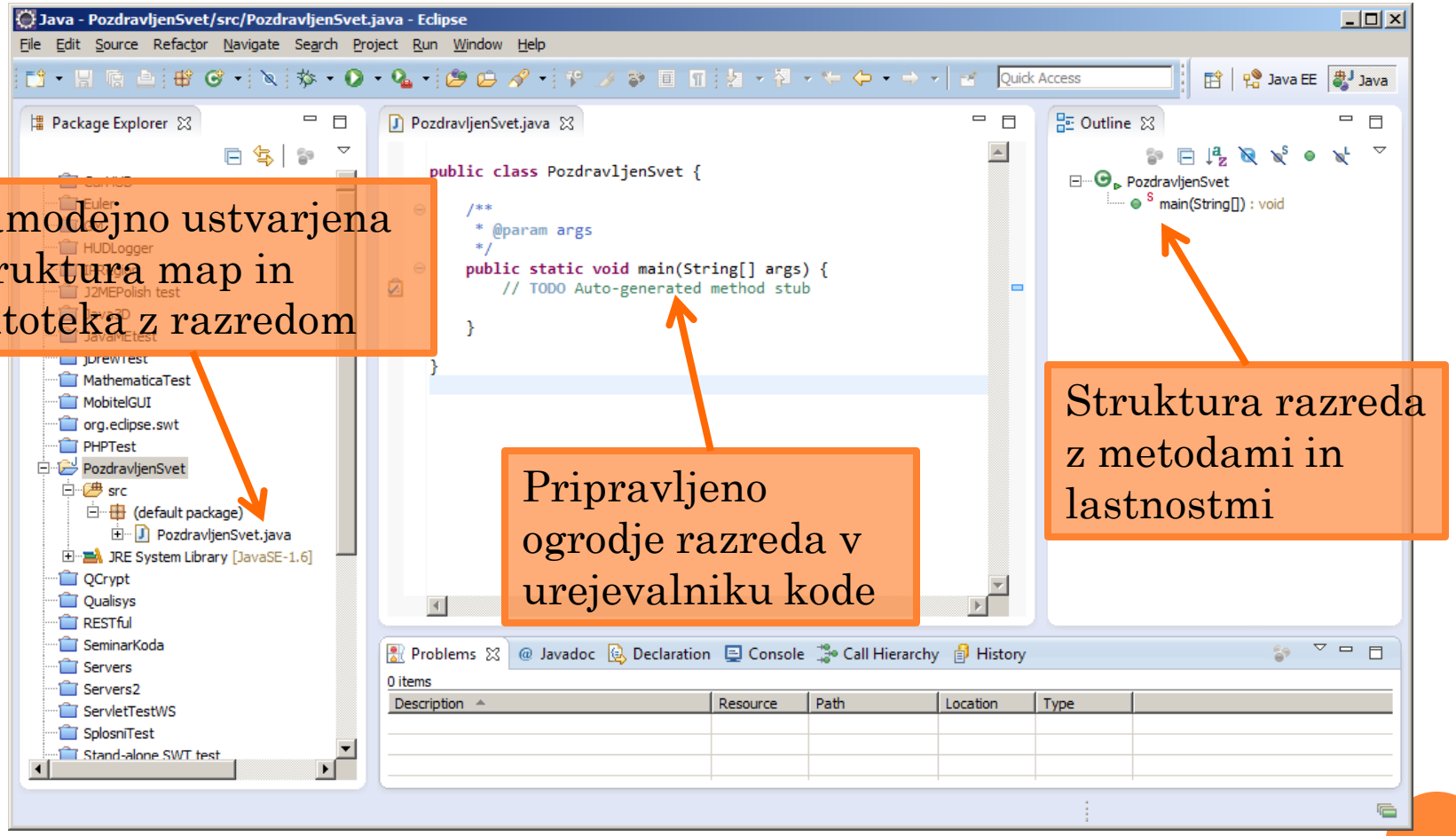


Izdelava javanskega razreda

- Razred je nekakšna predloga, iz katere so izpeljani posamezni objekti
- Desni klik na projekt → New → Class
- Če želimo, da bo razred vstopna točka aplikacije, moramo dodati metodo `main`



Izdelava javanskega razreda



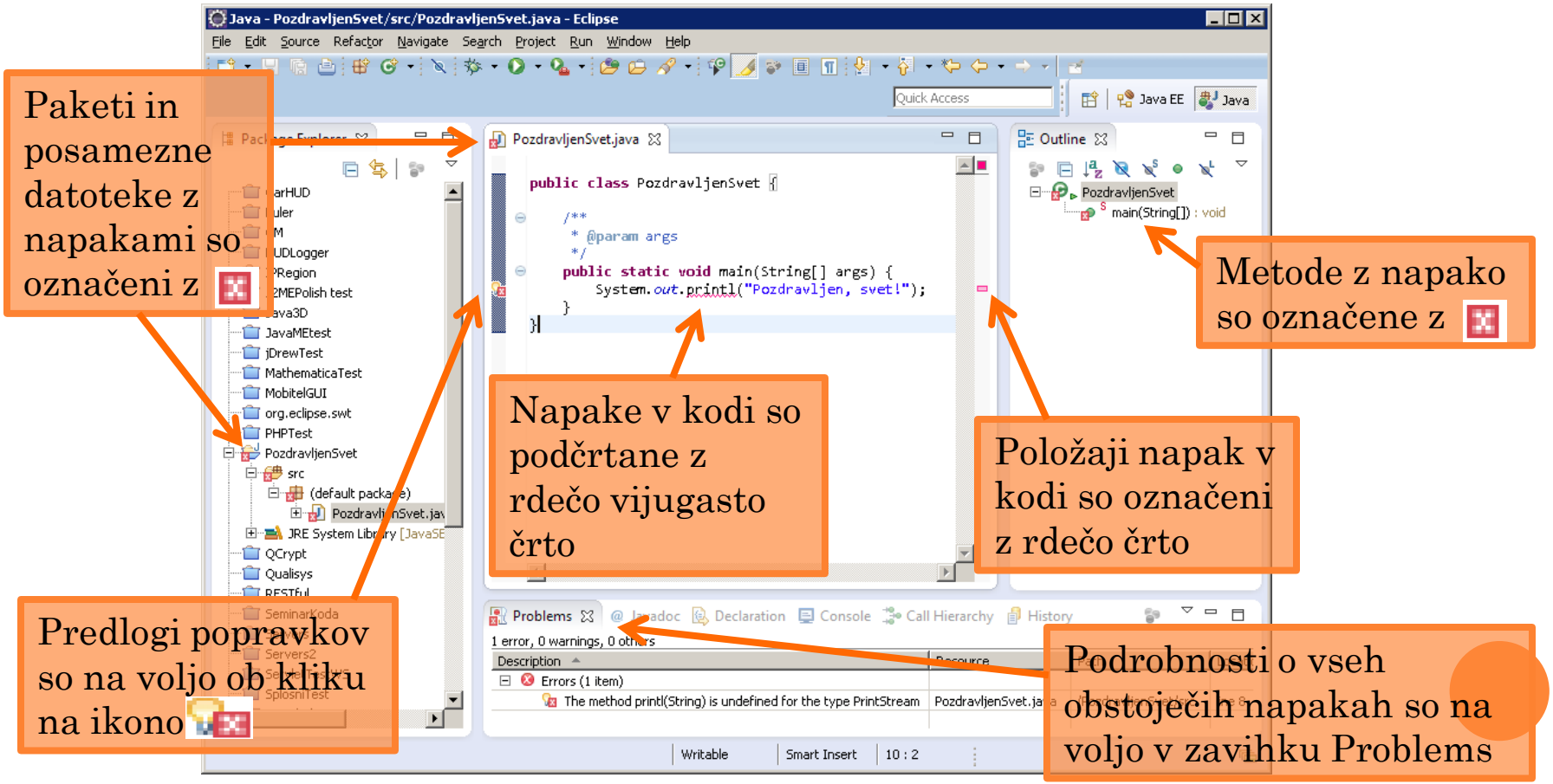
Prevajanje in poganjanje izvorne kode

- Eclipse samodejno prevede programsko kodo med samim pisanjem kode (možnost Project → Build automatically)
  - Napake lahko odpravljamo sproti
- Poženemo lahko kateri koli javanski razred, ki ima metodo `main`
  - ta metoda služi kot vstopno mesto aplikacije
- Javansko aplikacijo poženemo z desnim klikom na razred z metodo `main`, nato pa izberemo Run As → Java Application
- Rezultat izvajanja je viden v grafičnem uporabniškem vmesniku (če ga aplikacija ima) ali pa v konzoli

- Napišite program "Pozdravljen svet" in ga poženite
  - V konzolo lahko sporočila izpisujete z uporabo metode `System.out.println`

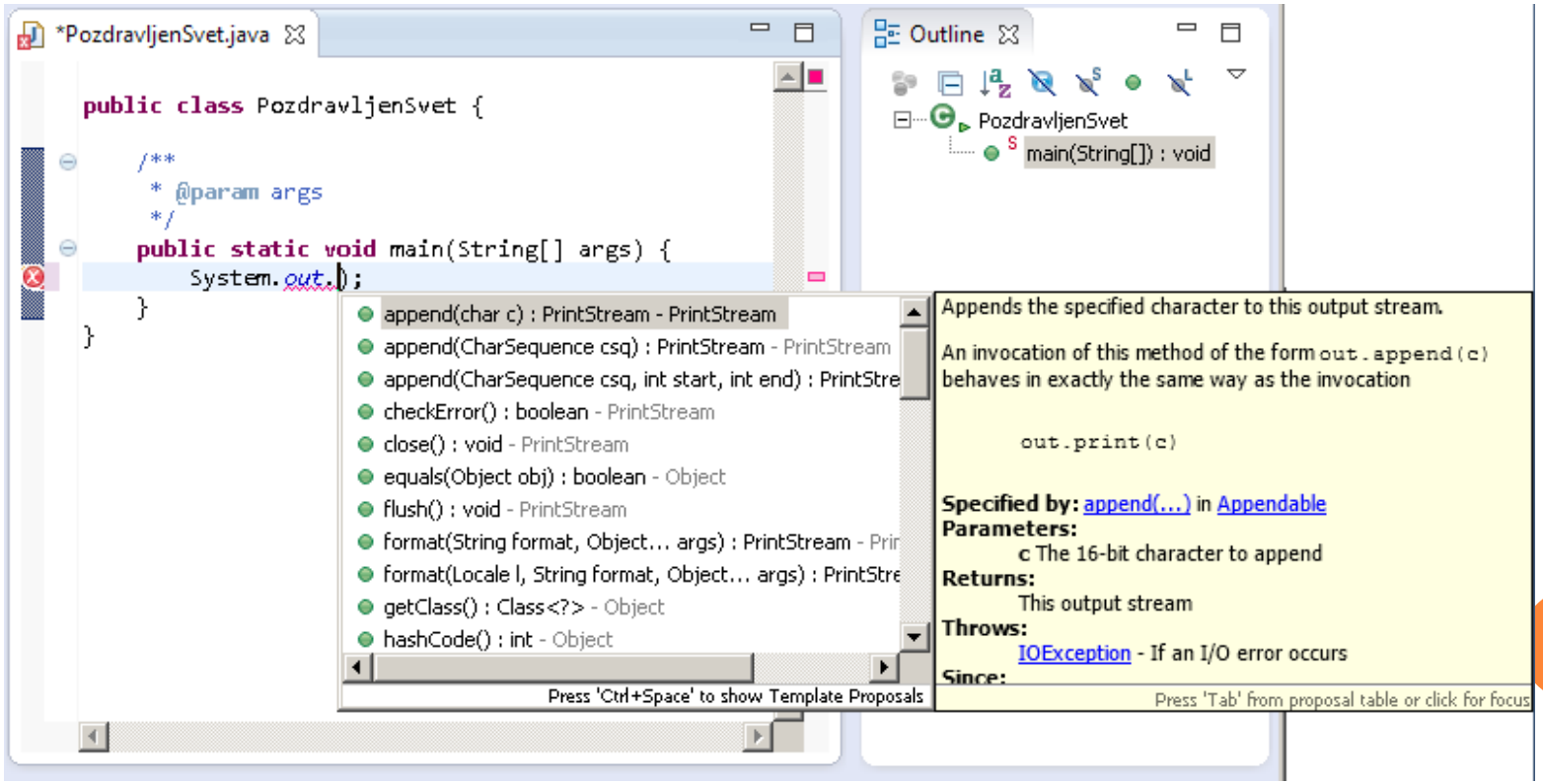
Napake v kodi

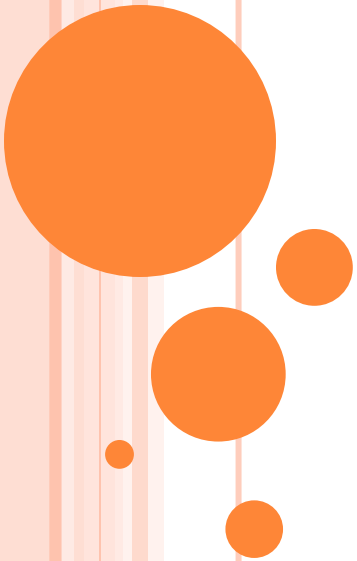
- Ker Eclipse samodejno prevede programsko kodo med samim pisanjem, lahko napake odpravljamo sproti



Pomoč pri pisanju programa

- Eclipse lahko ponudi možnosti, ki so nam na nekem mestu v izvorni kodi na voljo
  - ko na nekem mestu končamo s piko in počakamo trenutek
  - ali pritisnemo CTRL - presledek





OSNOVNI KONCEPTI PROGRAMSKEGA JEZIKA JAVA

Grega Jakus

15-10-2013

Naloge

- Pripravite okolje Eclipse, testni projekt in razred, v katerega boste pisali kodo
- V testni projekt uvozite pripravljeno kodo
- Pri študiju si pomagajte s sledečo literaturo
  - Uroš Mesojedec, Borut Fabjan, Java2: Temelji programiranja, Pasadena 2004
  - Spletni tečaji Jave

Temeljni elementi jezika

Izrazi, stavki in bloki

Metode

Simbolična imena

Ključne besede

Dobesedne vrednosti

Ločila in komentarji

Spremenljivke in podatkovni tipi

Operatorji

Izrazi, stavki in bloki

- **izrazi** = elementi programa, ki se izračunajo in vrnejo novo vrednost
- izraz (*expression*) =
  dobesedne vrednosti (*literals*) +
  spremenljivke (*variables*) +
  operatorji (*operators*) +
  klici metod (*method calls*)
- **stavek** (*statement*) = izraz, ki lahko nastopa samostojno
  - napovedni stavek (*declaration*)
  - kontrolni stavek (*control flow statement*)
- stavek se vedno konča s podpičjem;
- **blok** je skupina stavkov

NALOGA: V razredu Racunaj poiščite vsakega izmed zgoraj omenjenih konceptov

Racunaj.java

Metode

- Podprogrami, ki nadzorujejo obnašanje razreda/objekta
- Konvencija: metode pišemo z malo začetnico
  - primer: **`potenciraj`**
- Deklaracija metode
  - Imenu metode sledijo oklepaji (), v katerih naštejemo parametre metode ter njihove podatkovne tipe
  - pred imenom metode navedemo tip podatka, ki ga metoda vrača kot rezultat svojega delovanja
  - primer: `double potenciraj(int osnova, int stopnja)`
- Podatke iz metode "vrnemo" klicatelju metode s stavkom `return`.
  - primer: `return rezultat;`
- Če metoda ne vrača ničesar, potem jo označimo z `void`, stavek `return` pa v tem primeru ni potreben.

Racunaj.java

Primera metod

```
static double potenciraj(int osnova, int stopnja){

  double rezultat = Math.pow(osnova, stopnja);

  return rezultat;
}

static void izpisiRezultat(double rezultat){

  System.out.println(
      "Rezultat potenciranja je "+rezultat);
}
```

Racunaj.java

Metoda main

```
public static void main(String[] args){
   potenciraj(2,4);
}
```

- Metoda `main` je "vstopno mesto" aplikacije
  - Iz te metode kličemo ostale metode, ki sestavljajo aplikacijo
- Metoda `main` je vedno deklarirana kot `public static void main(String[] args)`
- Metoda `main` ne vrača ničesar (`void`)
  - Komu ali čemu pa bi sploh vračala?
- Metoda `main` sprejema en sam parameter `(String[] args)`
  - Polje znakovnih nizov, ki ustrezajo argumentom pri klicu aplikacije
  - Klic aplikacije v ukazni vrstici: `java` *`MojaAplikacija arg1 arg2 …`*
- Metoda `main` je javna (`public`)
  - mora biti, saj jo je namreč potrebno poklicati izven razreda, v katerem je definirana
- Metoda `main` je statična (`static`)
  - obstaja le ena različica te metode, ne glede na število objektov, ki jih izpeljemo iz razreda

Simbolična imena (identifiers)

- Pripišemo jih spremenljivkam, razredom, vmesnikom, paketom, metodam …
- Začenjajo se s črko, dolžina ni omejena
- Konvencije
  - z veliko začetnico poimenujemo samo razrede in vmesnike
  - vse ostale elemente poimenujemo z malo začetnico
  - konstante zapišemo s samimi velikimi znaki (`PI`)
  - če je simbolično ime sestavljeno iz več besed, le-te ločimo z velikimi začetnicami, razen pri konstantah, kjer uporabimo podčrtaje: `rezultatPotenciranja`, `mojaMetoda()`, `STEVILO_PI`
- Simbolična imena ne smejo biti ključne besede

Ključne besede (keywords)

- Elementi jezika, ki imajo natančno določeno vlogo
  - oznake primitivnih podatkovnih tipov (int, long,…)
  - zančne ključne besede (for, while, break, …)
  - pogojne ključne besede (if, else, switch, …)
  - ključne besede pri izjemah (try, catch, throw, …)
  - strukturne ključne besede (abstract, class, interface, …)
  - ključne besede dostopnosti in uskladitve (void, new, …)
  - preostale ključne besede (import, return, package, this, …)
- Eclipse jih obarva vijolično
- http://www.dummies.com/how-to/content/java-keywords.html

Dobesedne vrednosti (literals)

- prirejamo jih lahko podatkom osnovnih (primitivnih) tipov
  - cela števila
    - 42 (desetiški), 052 (osmiški), 0x2A (šestnajstiški zapis)
  - realna števila
    - decimalno ločilo je pika: 123.45, 1.2345e2f
  - logični vrednosti *true* in *false*
  - nedoločena vrednost *null*
  - znaki
    - znake zapisujemo med enojnimi narekovaji: ‘a’
    - na voljo je 16 bitna Unicode abeceda
  - nizi znakov
    - zapisujemo jih med dvojnimi narekovaji
      - “*to je niz*”
    - sestavljamo jih z operatorjem +
      - “*to*”+” “+”*je*”+” “+”*niz*”

Ločila in komentarji

- Programske stavke ločujemo s podpičjem **;**
- Sorodne programske stavke združimo v bloke z zavitimi oklepaji **{ }**
- Navadni oklepaji **( )** se uporabljajo za
  - zapisovanje parametrov metod
  - določanje vrstnega reda izračunov
- Če je parametrov metode več, uporabimo vejico **,**
- Oglate oklepaje **[ ]** uporabljamo za naslavljanje elementov v poljih
- Komentarji
  - `// komentar do konca vrstice`
  - `/* večvrstični komentar */`
  - `/** dokumentacijski komentar */`

Spremenljivke in podatkovni tipi

Skupine podatkovnih tipov
Napoved, vzpostavitev in doseg spremenljivke
Konstante
Primitivni podatkovni tipi
Sklicni podatkovni tipi

Skupine podatkovnih tipov

- Podatke v programu shranjujemo v spremenljivke
- Spremenljivke v objektno orientiranih jezikih lahko vključujejo tudi objekte
- Vsebino spremenljivke določa njen podatkovni tip
- Java pozna dve skupini podatkovnih tipov
  - primitivne podatkovne tipe, ki dejansko hranijo podatke
    - byte, short, int, long, float, double, char in boolean
  - sklicne podatkovne tipe, ki hranijo le naslov podatkov (sklic, referenco)
- Sklicu na nek podatek pravimo *kazalec*
  - java ne omogoča neposrednega dela s kazalci (kot npr. C++), namesto tega uporablja spremenljivke sklicnih podatkovnih tipov
- Kot že vemo (glej "Metode"), poleg spremenljivkam, prirejamo podatkovne tipe tudi
  - vrednostim, ki jih vračajo metode
  - parametrom, ki jih pošiljamo metodam

Napoved, vzpostavitev in doseg spremenljivke

- Napoved spremenljivke

```
podatkovni-tip imeSpremenljivke1 [, imeSpremenljivke2,…];
int a, b, c;
```

- Spremenljivko lahko napovemo v katerem koli bloku
- Spremenljivka je "vidna" od mesta napovedi do konca stavčnega bloka
- Pred uporabo moramo spremenljivki s prireditvenim stavkom prirediti vrednost

```
int a=3; //prirejanje (začetne) vrednosti ob napovedi spremenljivke
a=4;  //prirejanje (nove) vrednosti že napovedani spremenljivki
```

Naloga

- Kolikšna je vrednost spremenljivk na označenih mestih?

```
int a=0;

void karEnaMetoda(){
        int a=3;
        // a ?
        for(int i=0;i<4;i++)){
                int b=4;
                // a, b, i ?
        }
        // a, b, i ?
}

void seEnaMetoda(){
        // a ?
}
```

Konstante

- Java obravnava konstante kot "dokončne spremenljivke"
- Napoved je enaka napovedi spremenljivke, le da dodamo še ključno besedo **`final`**
- Vrednost konstante moramo določiti že ob napovedi, kasneje je ne moremo več spremeniti

```
final float PI = 3.14F;
```

- Pri poimenovanju konstant upoštevajte konvencije (glej "Simbolična imena")

Primitivni podatkovni tipi

Primitive Type	Size	Minimum Value	Maximum Value
char	16-bit	Unicode 0	Unicode $2^{16}-1$
byte	8-bit	-128	+127
short	16-bit	-2^{15} *(-32,768)*	$+2^{15}-1$ *(32,767)*
int	32-bit	-2^{31} *(-2,147,483,648)*	$+2^{31}-1$ *(2,147,483,647)*
long	64-bit	-2^{63} *(-9,223,372,036,854,775,808)*	$+2^{63}-1$ *(9,223,372,036,854,775,807)*
float	32-bit	32-bit IEEE 754 floating-point numbers	
double	64-bit	64-bit IEEE 754 floating-point numbers	
boolean	1-bit	`true` or `false`	
void	-----	-----	-----

- Razlikujejo se po razponu vrednosti – količini pomnilniškega prostora
- Če pri navajanju dobesedne vrednosti uporabimo decimalno ločilo (piko), bo prevajalnik obravnaval število kot double
- Če želimo uporabiti tip float, dodamo za število znak f
- Podobno velja za kombinacijo int (zapis brez pike) in long (dodamo L)

Ovijanje celoštevilčnih tipov

Kaj izpiše spodnja koda ?

```
byte krog=127;
krog++;
System.out.println(krog);
```

- Prekoračitvi obsega spremenljivke pravimo *ovijanje*
- **Ovijanje je logična napaka v programu**
  - razen če ga izkoriščamo namerno
- Prevajalnik ali izvajalno okolje ne javljata napake v primeru ovijanja, zato moramo sami poskrbeti, da ne prekoračimo obsega podatkovnega tipa!!

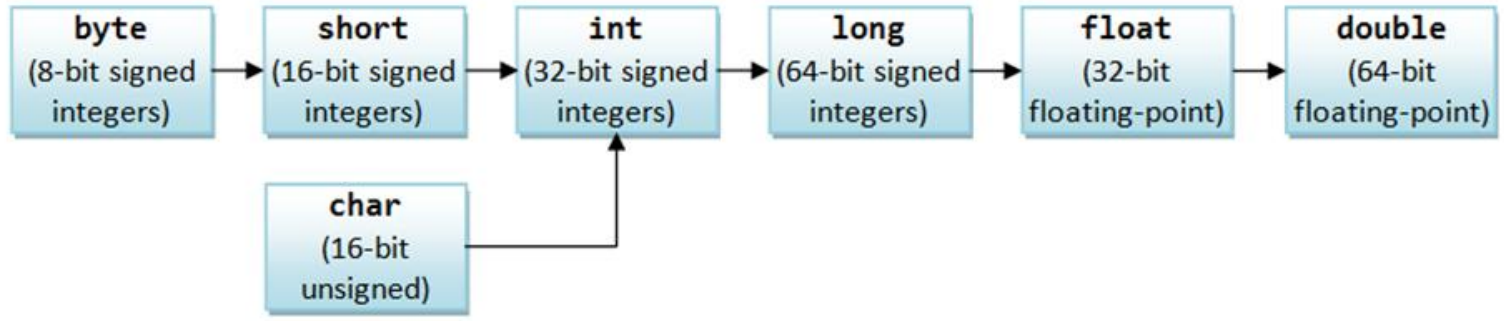
Širitev spremenljivk (implicit type casting)

- Če v izrazu uporabimo dve ali več spremenljivk različnih obsegov (tipov), se
  1. vse spremenljivke najprej razširijo na največji uporabljen obseg in
  2. nato šele uporabijo v izračunu
- Primeri
  - byte, long → long
  - float, double → double
  - celo št., realno št. → realno št.

- Kaj izpiše `System.out.println(5/2)`?
- Kaj izpiše `System.out.println(5/2.0)`?

Vsiljeno pretvarjanje tipov (explicit type casting)

- Kadar pretvorba tipa ne povzroči zmanjšanja natančnosti, se izvede samodejno (glej "Širitev spremenljivk")



Orders of Implicit Type-Casting for Primitives vir: http://www.ntu.edu.sg/home/ehchua/programming/java/images/JavaBasics_ImplicitTypeCastingPrimitives.png

- Vsiljeno pretvarjanje tipov dosežemo z zapisom novega tipa v oklepaju:

```
double x = 12.9412;
int i= (int) x;
```

- Pri vsiljenem pretvarjanju tipov imamo lahko zaokrožitveno napako
- Katero vrednost hrani spremenljivka i v zgornjem in katero v spodnjem primeru?

```
int i = (int) Math.round(x);
```

Operatorji primitivnih tipov

Operator Precedence

Operators	Precedence
postfix	`expr++ expr--`
unary	`++expr --expr +expr -expr ~ !`
multiplicative	`* / %`
additive	`+ -`
shift	`<< >> >>>`
relational	`< > <= >= instanceof`
equality	`== !=`
bitwise AND	`&`
bitwise exclusive OR	`^`
bitwise inclusive OR	`|`
logical AND	`&&`
logical OR	`||`
ternary	`? :`
assignment	`= += -= *= /= %= &= ^= |= <<= >>= >>>=`

Ovijalni razredi (wrapper type)

- Vsakemu primitivnemu tipu je dodeljen *ovijalni razred*,
  - ki hrani podatek primitivnega tipa,
  - obenem pa ponuja metode za delo s primitivnimi tipi
    - primerjanje, pretvorbe, skrajne vrednosti primitivnega tipa itd.
- Ovijalni razredi sodijo v skupino sklicnih podatkovnih tipov

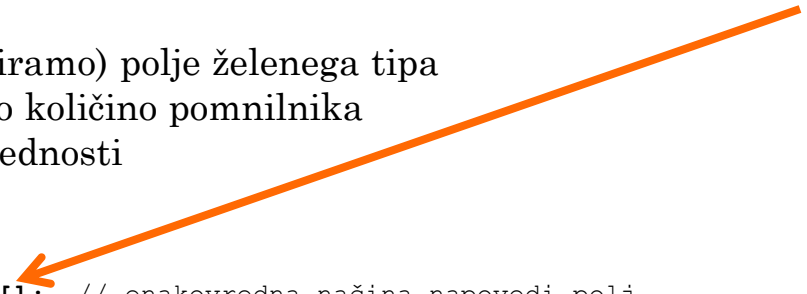
Primitive Type	Wrapper Type
char	Character
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
void	Void

Sklicni podatkovni tipi

- Predstavljajo sestavljene podatkovne strukture
- Sklicne spremenljivke ne shranjujejo podatkov, ampak le njihov naslov v pomnilniku
- Za dodeljevanje in sproščanje pomnilnika skrbi javanski navidezni stroj
- V javi so vsi podatki, razen spremenljivk primitivnih tipov, sklicnega tipa
  - polja
  - znakovni nizi
  - ostali objekti

Polja (arrays)

- Polja so zbirke podatkov istega tipa (primitivnega ali sklicnega)
- Polje napovemo s parom oglatih oklepajev ob tipu ali simboličnem imenu polja
- Uporaba polj
  - napovemo (deklariramo) polje želenega tipa
  - zasežemo potrebno količino pomnilnika
  - v polje vpišemo vrednosti
- Primeri:



napoved polja

```
int[] polje1; int polje2[];  // enakovredna načina napovedi polj
String[] pozdrav;    //napoved polja objektov (vrste String)
int[][] matrika;     //polje polj (večdimenzionalno polje)
```

hkratna napoved in zaseganje pomnilnika

```
float koordinate[]=new float[3];
```

hkratna napoved, zaseganje pomnilnika in prirejanje začetnih vrednosti

```
float koordinate[]={4.5, 1.22, -4.3};        //Prevajalnik sam izračuna dolžino polja!
int matrika[][]={{4, 1, -4},{1,0,0},{1,2,-4}};
```

naknadno prirejanje vrednosti posameznim elementom polja

```
koordinate[0]=2.3;                //Indeks polja se vedno začne z 0!!
koordinate[1]=22.43;
koordinate[2]=0.0;
```

Polja

- Ko polju enkrat določimo dolžino (zasežemo pomnilnik), te ne moremo več spremeniti
- Dolžino polja ugotovimo prek lastnosti length:

```
int polje={1,2,3};
int velikostZgornjegaPolja=polje.length;
```

- Do posameznega elementa polja dostopamo z njegovim indeksom v oglatih oklepajih
  - Indeks polja se vedno začne z 0
- Metode za delo s polji najdemo v razredu java.util.Arrays

- Sprehod skozi polje

```
for(int i=0; i<polje.length;i++){
  //do trenutnega elementa polja lahko dostopamo
  //preko izraza polje[i]
}
```

Znakovni nizi

- Niso primitivni tip, temveč objekti, izpeljani iz razreda String
- Zapišemo jih med dvojnima narekovajema:

```
"to je nek niz"
```

- Dolžino niza ugotovimo s pomočjo metode `length`

```
"to je nek niz".length();
```

- Nize združujemo z uporabo operatorja +

```
"to" + " " + "je" + " " + "en" + " " + "niz"
```

- Združevanje s podatki primitivnega tipa

```
"temperatura je " + 25.5f +" stopinj"
```

- Primerjanje nizov
  - Znakovnih nizov ne primerjamo z operatorjem == temveč v ta namen uporabimo metodi `equals` ali `equalsIgnoreCase`
  - Primer: `if(prviNiz.equals(drugiNiz))`
- Razred String ponuja nabor metod za lažje delo z nizi
  - dostop do delov niza, primerjanje, zamenjava, sprememba velikosti črk, odstranitev presledkov itd.

Nadzor izvajanja

Kontrolni stavki

- Odločitvena stavka
  - if in switch stavek
- Ponavljalni stavki
  - for, while, do-while
- Razvejani stavki
  - break, continue, return

Stavek if

```
if(pogoj){
   //telo
} else{
   //telo
}
```

```
if(x>=0){
    int koren=Math.sqrt(x);
}

if(a==3){
        //koda
}
else{
        //koda
}

if(niz.equals("tako se primerja pa znakovne nize")){
        //koda
}
```

Stavki for, break in continue

for (*začetni-stavek*; *pogoj*; *ponovitveni-stavek*) {
    //telo
}

```
for(int i=0; i<polje.length; i++){
   // če je i sodo število, bomo izvajanje zanke prekinili
   // in nadaljevali z naslednjim i
   if(i%2==0) continue;

   // če i-ti element polja vsebuje število 13,
   // predčasno zaključimo izvajanje zanke
   if(polje[i]==13) break;

   // poljubna programska koda…
}
```

Naloge

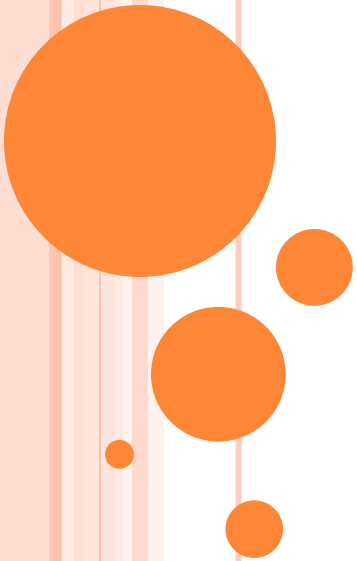
- Seštejte prvih sto števil, večjih od 500, ki so deljiva s tri a ne s pet.
- Napišite program, ki obrne elemente polja nizov, v vsakem nizu pa še posamezne znake.
  - Primer:

    {“To”, “je”, ”polje”, “nizov”}

    {“vozin”, ”ejlop”, “ej”, “oT”}

Stevila.java, ObrniPoljeNizov.java



OBJEKTNO PROGRAMIRANJE V JAVI

Grega Jakus

19-10-2013

Objektno (predmetno) programiranje

- Cilj objektnega programiranja je zgraditi program iz standardnih komponent, ki jih je mogoče ponovno uporabiti in na ta način zmanjšati kompleksnost programiranja
- Objekti (predmeti) posnemajo predmete iz realnega sveta
  - imajo stanje, shranjeno v spremenljivkah = lastnosti objekta
  - imajo obnašanje, določeno preko metod = objektu lastne funkcije
- Programski objekti torej združujejo podatke in postopke za njihovo obdelavo
- Program v javi je navadno sestavljen iz več objektov, ki med seboj komunicirajo preko sporočil – klicev izpostavljenih (javnih) metod
- Pri objektnem programiranju je zelo pomembno predhodno načrtovanje programa
  - razdelitev programa (problema) na posamezne gradnike (razrede) in določitev njihovih lastnosti in obnašanja

Razred (class)

- Razred je načrt (prototip) sorodnih objektov
  - Razred vsebuje skupne značilnosti sorodnih objektov
  - Na podlagi razreda lahko izdelamo poljubno število različnih objektov
  - Primer:
    - razred: načrt za letalo Airbus A380
    - objekti: konkretna letala izdelana na podlagi načrta
- Razred določa skupno obnašanje in začetno stanje vseh, iz njega ustvarjenih objektov
- Posamezen razred je samostojni sklicni podatkovni tip
- V javi navadno vsak razred zapišemo v ločeno datoteko

Paket (package)

- Paketi so skupine sorodnih razredov
- Ime paketa je hierarhično urejeno
  - podobno kot URL naslov, le obrnjena smer navajanja nivojev
  - posamezni nivoji so ločeni s pikami
- Primera
  - java.util
  - si.uni_lj.fe.seminarTK
- Polno ime razreda je sestavljeno iz imena paketa, v katerega je razred uvrščen, in imena razreda
  - java.util.Date
  - java.util.* // označuje vse razrede, ki so v paketu java.util
- Razrede v javansko kodo uvozimo s pomočjo rezervirane besede import
  - import java.util.Date;
  - import java.util.*;

V okviru delovnega projekta ustvarite paket si.uni-lj.fe.seminarTK!
Naša koda bo od zdaj naprej v omenjenem paketu!

Iskanje paketov in razredov med izvajanjem

- Ko prevajalnik potrebuje nek razred, mora vedeti, kje ga poiskati
  - Hierarhija v imenu paketa se odrazi tudi v datotečnem sistemu
  - Ime razreda ustreza datoteki z izvorno kodo
- Datoteka bo v mapi, katere relativno pot določa ime paketa, v katerega je razred uvrščen
  - primer: relativna pot do datoteke z razredom Student, ki je del paketa si.uni_lj.fe.seminarTK bo si/uni_lj/fe/seminarTK/Student.java
- Izhodišča relativnih poti v datotečnem sistemu so lahko različna
  - trenutna delovna mapa (v delovnem prostoru Eclipsa)
  - mapa, v kateri je nameščeno izvajalno okolje
  - poljubna ostala izhodišča
- Vsa izhodišča, v katerih išče razrede prevajalnik, so zapisana v sistemski spremenljivki CLASSPATH
- V Eclipsu CLASSPATH razširja t.i. Build Path

Objekt (object)

- Objekt je konkretna različica razreda, ki obstaja le v pomnilniku v času izvajanja programa
- Vsi objekti, ki so ustvarjeni iz istega razreda, imajo enako obnašanje in lastnosti, a imajo lahko različna stanja = vrednosti lastnosti
- Objekt ustvarimo iz razreda z ukazom `new`:

```
new NemškiOvčar();
```

- Spremenljivki, ki ji priredimo nek razred kot tip, pravimo predmetna spremenljivka

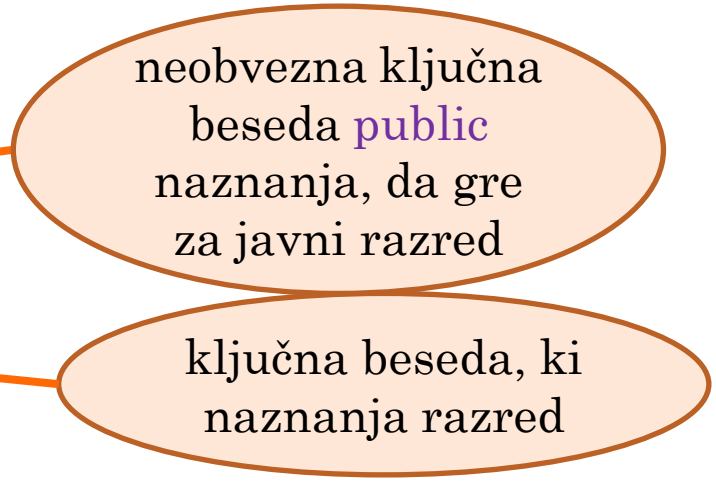
```
NemškiOvčar lassie = new NemškiOvčar();
```

- Ko objekt ustvarimo, lahko kličemo njegove metode

```
lassie.zalajaj();
```

Zgradba razreda

- Zapis razreda




```
[public] class ImeRazreda{
  //vsebina razreda
}
```

- Vsebina razreda so lastnosti, konstruktorji in metode
  - Lastnosti so spremenljivke, v katerih je shranjeno stanje objekta kot različice razreda
  - Konstruktorji vzpostavijo začetno stanje novega objekta
  - Javne metode služijo dostopanju in spreminjanju stanja objekta "od zunaj", zasebne metode pa vključujejo funkcionalnost objekta, ki od zunaj ni dostopno
- Primer: koda Oseba.java

Oseba.java

Lastnosti (properties)

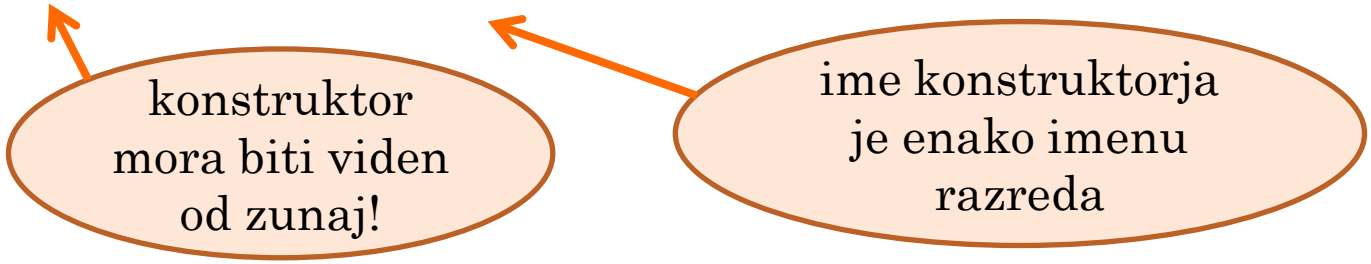
- Lastnosti so spremenljivke, ki predstavljajo stanje objekta
- Spremenljivke stanja so dostopne vsem metodam objekta
- Dostopnost za druge objekte lahko določimo sami:
  - stanje objekta skrijemo s **private**:
    ```
    private int starost=45;
    ```
  - stanje izpostavimo s **public**
    ```
    public int ocenaNaIzpituSeminarIzNPOvTK=10;
    ```
  - **protected** omeji dostopnost na
    - vse razrede paketa in
    - vse razrede, ki so izpeljani iz aktualnega razreda
  - če ne navedemo nobene ključne besede, omejimo dostopnost na vse razrede v paketu, v katerem je aktualni razred
- Začetno stanje objekta, vzpostavimo s konstruktorjem

Oseba.java

KONSTRUKTOR (CONSTRUCTOR)

- Konstruktor je posebna "metoda"
  - izvede se samodejno, ko ustvarimo objekt
  - ne moremo ga poklicati kot pokličemo navadne metode
  - ne vrača vrednosti
- Objekt ima lahko nič, enega ali več konstruktorjev
- Če ima razred več konstruktorjev, se ti razlikujejo v podpisu
  - podpis = ime metode in seznam parametrov
  - izbere se tisti konstruktor, ki ustreza podanim parametrom pri ustvarjanju objekta
- Podpis konstruktorja:

```
public ImeRazreda(/*parametri kot v metodah*/)
```

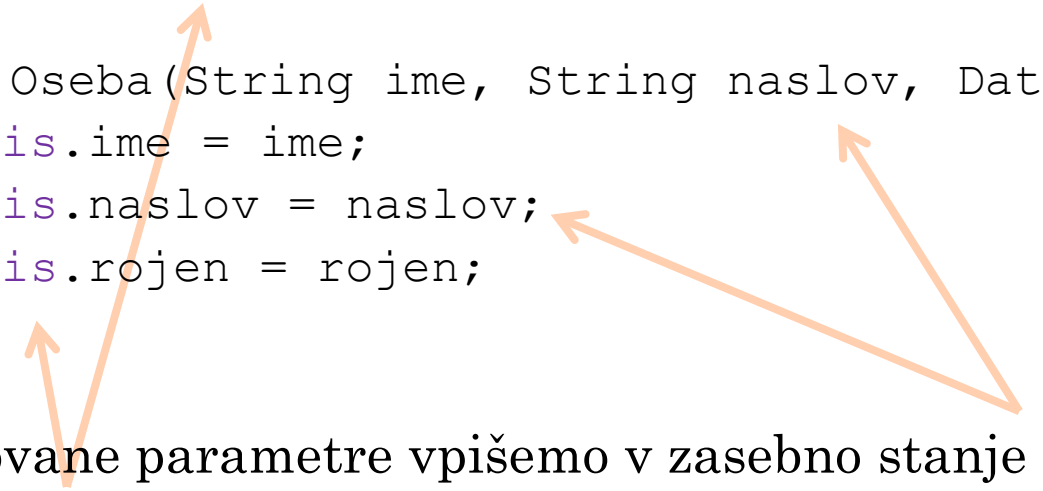


Oseba.java

Vzpostavitev stanja s konstruktorjem

```
public class Oseba{
   private String ime;
   private String naslov;
   private Date rojen;

   public Oseba(String ime, String naslov, Date rojen){
        this.ime = ime;
        this.naslov = naslov;
        this.rojen = rojen;
   }
}
```



- Posredovane parametre vpišemo v zasebno stanje objekta
- Beseda this pomaga razrešiti konflikt med imeni parametrov pri klicu konstruktorja in istoimenskimi spremenljivkami stanja
- Rezervirana beseda this predstavlja aktualni objekt (različico razreda)

Oseba.java

METODE (METHODS)

- Dostopnost: velja podobno, kot pri dostopnosti lastnosti razreda
  - public, private, protected, nič od tega
- Parametri metod
  - se prenašajo po vrednosti in ne po referenci (sklicu)
  - metode uporabljajo kopije posredovanih podatkov
    - pri primitivnih tipih se spreminjanje posredovanih podatkov znotraj metode ne odrazi na originalnih podatkih zunaj metode
    - ker kopija sklicne spremenljivke kaže na iste podatke kot original, **spreminjanje podatkov sklicnega tipa spreminja vrednost podatkov, na katere se sklicuje posredovana spremenljivka (objekt)**
      - Npr. lastnost "rojen" v Oseba.java

Oseba.java

Statični člani razreda (static members)

- Za uporabo statičnih članov ni potrebno ustvariti novega objekta, saj so statični člani **del razreda** in kot taki prek razreda tudi dostopni vsem različicam (objektom)
- Primeri
  - metoda `main`
  - `Math.Cos();` // Vse metode razreda Math so statične
  - `Math.E;` // podobno kot za metode in spremenljivke velja tudi za konstante
- Statične metode ne morejo uporabljati predmetnih spremenljivk, ki niso statične
  - razen če statične metode same ustvarijo objekt in se z njegovo pomočjo sklicujejo na njegovo stanje
  - primer: metoda `main`
- Statične spremenljivke imajo doseg razreda
  - na voljo so **v enem izvodu** vsem objektom, ustvarjenih iz razreda
  - na voljo so tudi, če iz razreda ne ustvarimo nobenega objekta
  - primer: spremenljivka, ki vsebuje števec objektov, ustvarjenih iz razreda Oseba.java

Oseba.java

Zaščita stanja objekta (encapsulation)

- Pomembno je imeti popoln nadzor nad stanjem objekta
  - stanje zato shranimo v zasebnih (private) spremenljivkah, ki niso *neposredno* dostopne izven objekta
  - dostop do stanja objekta omogočimo *posredno* preko javnih metod
    - za vsako lastnost objekta naredimo ločeni metodi za branje in/ali spreminjanje lastnosti
  - izpostavimo le tista stanja, za katera je to nujno potrebno
- Javne člane razreda imenujemo *javni vmesnik* razreda
  - javni vmesnik omogoča, da je nek objekt mogoče vedno uporabiti na enak način
    - (notranje) delovanje objekta se lahko spremeni, a njegova uporaba ostane enaka
  - pomembno je, da ostane javni vmesnik enak pri vseh različicah razreda, v nasprotnem primeru programi, ki razred uporabljajo, ne bodo več delovali

Oseba.java

Metode stanja ("Getters and setters")

Metode stanja zunanjemu svetu omogočijo branje in spreminjanje stanja objekta

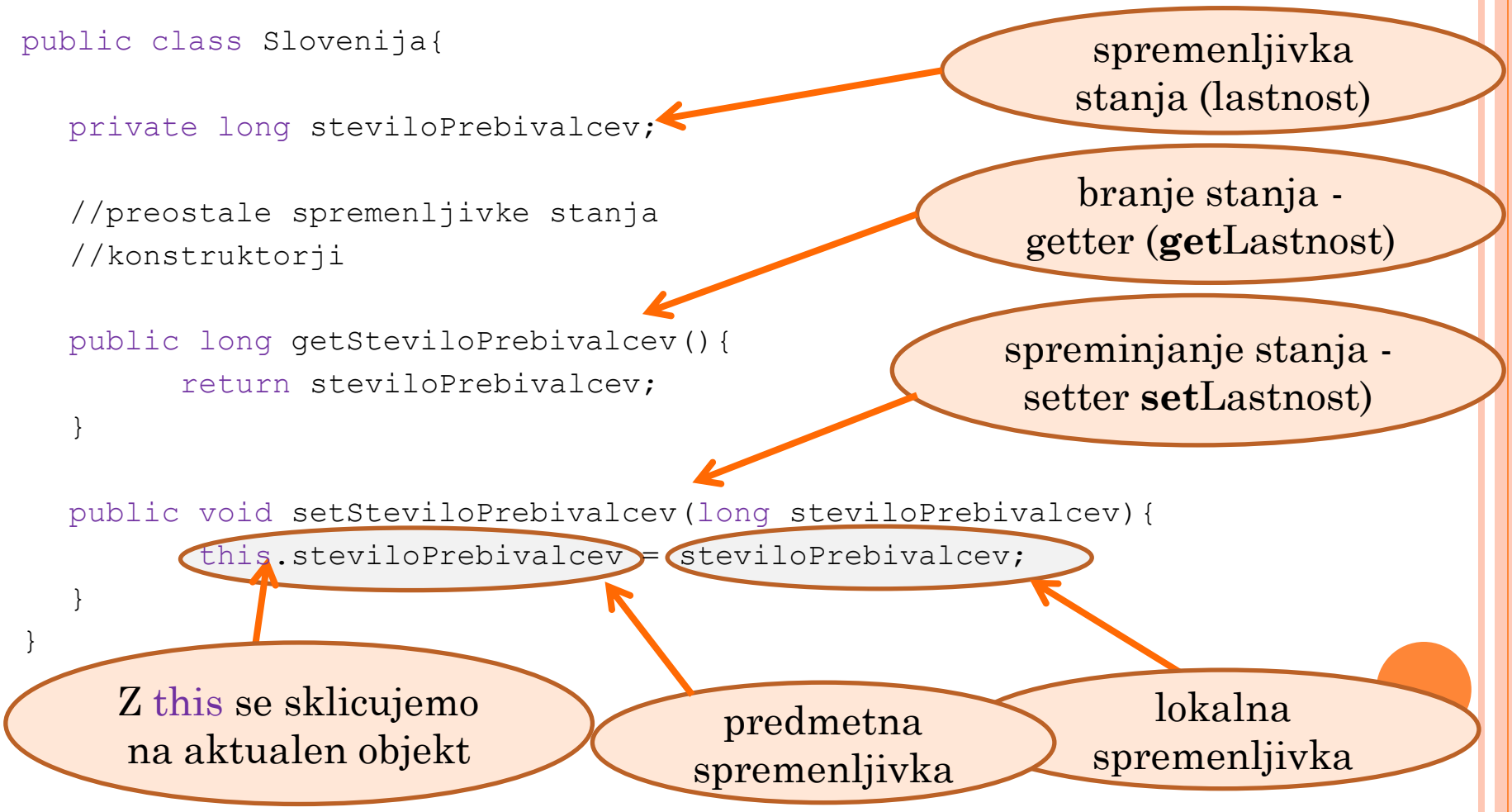
```
public class Slovenija{

   private long steviloPrebivalcev;

   //preostale spremenljivke stanja
   //konstruktorji

   public long getSteviloPrebivalcev(){
         return steviloPrebivalcev;
   }

   public void setSteviloPrebivalcev(long steviloPrebivalcev){
         this.steviloPrebivalcev = steviloPrebivalcev;
   }
}
```



Dostop do spremenljivk sklicnih tipov

- Spomnimo se: *„ker kopija sklicne spremenljivke kaže na iste podatke kot original,* ***spreminjanje podatkov sklicnega tipa spreminja vrednost podatkov, na katere se sklicuje posredovana sklicna spremenljivka“***
- Če v “getterju” vrnemo spremenljivko sklicnega tipa (objekt), potem dejansko vrnemo sklic na to spremenljivko

- Če vrnjen objekt omogoča spreminjanje svojega stanja, potem mu je mogoče stanje spremeniti tudi izven “matičnega” objekta, v katerem sam predstavlja stanje.

- Kadar v zasebnem stanju objekta uporabljamo objekte, ki jim lahko spremenimo stanje, teh objektov ne vračamo neposredno, ampak vrnemo njihovo kopijo – nov objekt z enakim stanjem!

```
//NAROBE:
public Date getRojen(){
  return rojen;
}
```

```
//PRAVILNO:
public Date getRojen(){
  return new Date(rojen.getTime());
}
```

```
Date dr = oseba.getRojen();
dr.setTime(/*vpišemo nov datum*/);
```

Oseba.java

DEDOVANJE

Dedovanje (inheritance)

- Dedovanje je gradnja novih razredov na podlagi že izdelanih razredov
- Nov razred je izpopolnjena različica predhodnika, primerna za uporabo v specifičnih okoliščinah
- Dedovanje omogoča ponovno uporabo kode
- Dedovanje naznanimo z besedo extends:

```
public class Student extends Oseba {
  /*telo razreda*/}
```

- Dedovanje preprečimo, če razred označimo kot final
  - Preprečimo lahko tudi dedovanje samo posameznih metod

Izpopolnjevanje

- Novi razred "podeduje" stanje in metode nadrejenega (podedovanega) razreda, poleg teh pa ima svoje posebno stanje in metode

```
public class Oseba{
  private String ime, naslov;
  private Date rojen;

/* konstruktor, metode stanja,
ostale metode*/
}
```

```
public class Student extends Oseba{
  private String fakulteta;
  private int vpisnaSt, letnik;

/* konstruktor, metode stanja,
ostale metode*/
}
```

- Razred Student podeduje lastnosti razreda Oseba, prav tako pa lahko uporabljamo na objektu tipa Student tudi metode razreda Oseba (npr. getNaslov, setNaslov itd)
- Lastnosti in metode, ki so specifične za razred Student, so na voljo le v objektih, ustvarjenih iz tega razreda

Student.java

Povožene metode (Overridden methods)

- Potomci lahko posamezne metode dedovanega razreda prilagodijo svojim potrebam – jih "povozijo"
- Povozimo lahko tudi konstruktorje
- Povoziti ne moremo statičnih metod in metod, označenih s ključno besedo final

```
public class Oseba{
  private String ime;

  public String getIme(){
    return ime;
  }
}
```

```
public class Student extends Oseba{

   public String getIme(){
     return "Študent"+super.getIme();
  }
}
```

spremenjena (povožena) metoda getIme

s pomočjo rezervirane besede super dostopamo do povožene metode dedovanega razreda

Student.java

Mnogoličnost (polymorphism) in vsiljeno prirejanje tipov (type casting)

- Če je objekt ustvarjen iz razreda, ki deduje, lahko takemu objektu po potrebi izberemo podatkovni tip dedovanega (splošnejšega) razreda:
  - Če razred Student deduje razred Oseba, lahko uporabimo
    ```
    Oseba jurcek = new Student();
    ```
- Vsiljeno prirejanje tipov
  - pride v poštev pri klicanju metod dejanskega tipa (ne splošnejšega)
  - podobno vsiljenemu prirejanju tipov pri primitivnih spremenljivkah
  - Primer: objekt jurcek je dejanskega tipa Student, a ker smo mu določili splošnejši tip Oseba, ne moremo direktno uporabiti specifičnih metod razreda Student
  - Lahko pa uporabimo vsiljeno prirejanje tipov
    ```
    ((Student)jurcek).getVpisnaSt();
    ```
- Z operatorjem instanceof lahko ugotavljamo, če je podatkovni tip nekega objekta izbran razred
  - izraz `jurcek instanceof Student` vrne true ali false

Student.java

Abstraktni razred (abstract class)

- Zelo splošni razredi so lahko v praksi preveč splošni za praktično uporabo in jih je za to potrebno natančneje določiti
- Primer: nihče ni le "oseba" temveč je lahko predstavljen bolj konkretno, npr. kot "študent", "profesor", "gospodinja" ali "tajkun"
- Abstraktni razredi so splošni razredi, ki povzemajo le skupne značilnosti neke skupine razredov, ustvarjanje objektov neposredno iz njih pa ni uporabno, saj v praksi želimo bolj izpopolnjene različice – potomce abstraktnih razredov
- Abstraktni razred naznanimo z uporabo besede abstract
  - Razred je abstrakten, če ima vsaj eno abstraktno metodo (označeno z abstract)
- Abstraktne metode so le najavljene in nimajo vsebine
  - Izvedba metode je prepuščena potomcem razreda
- Ker so abstraktni razredi nepopolni, iz njih ne moremo ustvariti objektov
  - Lahko pa služijo kot splošnejši podatkovni tip pri izkoriščanju mnogoličnosti

```
abstract class OlikanaOseba{
   //ostala vsebina razreda
   public abstract String predstaviSe(); //najava metode brez izvedbe
}
```

OlikanaOseba.java

Naloge

- Dokončajte razred OlikanStudent, tako da bo pravilno dedoval razred OlikanaOseba

- Razred OlikanStudent vsebuje vsebino, ki smo jo že zasledili v razredu Student.

  Ali lahko dosežete, da bo razred OlikanStudent poleg razreda OlikanaOseba dedoval tudi razred Student?

OlikanaOseba.java, OlikanStudent.java

VMESNIK (INTERFACE)

- Vmesniki so abstraktni razredi, ki imajo vse metode abstraktne
- Vmesnik najavimo z besedo interface, ki nadomesti besedo class

```
public interface ZnanjeStudentaTK{
    public boolean nastejSlojeOSIModela();
    public boolean nastejSlojeTCP_IPModela();}
```

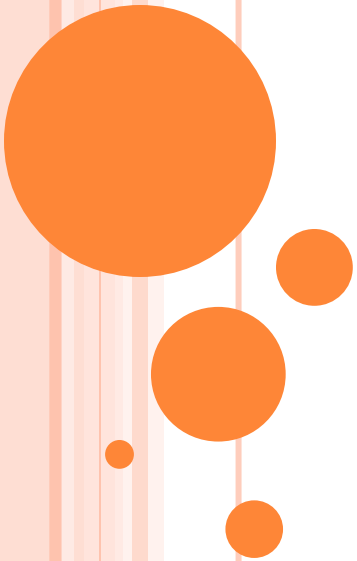
- Vmesnik je pogodba, ki določa, kaj morajo razredi narediti, ne pa kako naj to naredijo
  - Če razred naredi, kar narekuje vmesnik, pravimo, da razred "izdela" vmesnik
  - Razred, ki izdela nek vmesnik, izpolnjuje le *formalne* zahteve, potrebne za izpolnitev naloge
  - Da razred pravilno izpolnjuje zahteve, poskrbimo z izdelavo vsebine predpisanih metod v skladu s podano dokumentacijo vmesnika (API)
- Razred lahko izdela več vmesnikov
- Da razred zadošča pogodbi, določeni z vmesnikom, naznanimo z uporabo besede implements in imenom vmesnika

```
public class DiplomantUNITK extends Student implements ZnanjeStudentaTK
```

ZnanjeStudentaTK.java

Naloge

- Načrtujte vmesnik, ki služi kot ogrodje za opis različnih vozil. Vmesnik naj omogoča modeliranje osebnega avtomobila, motornega kolesa in tovornjaka.
- Izberite si eno izmed vozil in ga modelirajte z razredom, ki izdela načrtovani vmesnik.
- Preizkusite delovanje razreda



Java in podatkovne zbirke

Grega Jakus

4-11-2013

Java in podatkovne zbirke

- Java omogoča povezovanje s podatkovnimi zbirkami po standardu JDBC
  - Neodvisnost programske kode od podatkovnega strežnika in izvedbe podatkovne zbirke
- Za uporabo podatkovne zbirke moramo imeti
  - vir podatkov (npr. podatkovni strežnik)
    - MySQL, Oracle, DB2, Microsoft SQL server …
  - možnost vzpostavitve povezave z virom (npr. v obliki gonilnikov in omrežne povezave)
    - Potrebujemo gonilnik razvijalca strežnika za javansko platformo
    - Uporaba gonilnika je mogoča preko upravnika gonilnikov JDBC

Uporaba podatkovne zbirke

- Podatke iz podatkovne zbirke tipično uporabljamo na sledeči način
  1. Najprej preko ustreznega gonilnika ustvarimo povezavo s podatkovno zbirko
  2. Prek ustvarjene povezave dostopamo do podatkov s pomočjo jezika SQL
  3. Podatke obdelamo in jih s pomočjo jezika SQL po potrebi shranimo nazaj v zbirko podatkov

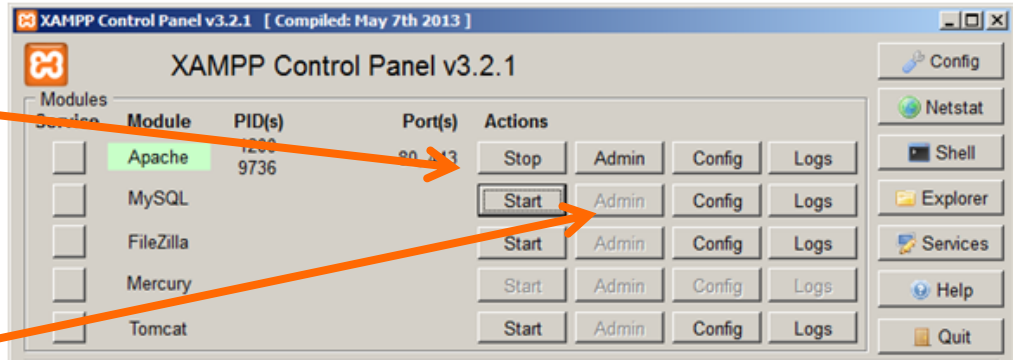
Jezik SQL

- Jezik SQL omogoča pridobivanje podatkov iz podatkovne zbirke in vstavljanje novih podatkov vanjo
- Poizvedbeni stavki
  - se pričnejo z SQL ukazom SELECT
  - sprožimo jih z metodo `executeQuery`
- Stavki, ki spremenijo vsebino v zbirki
  - se pričnejo z SQL ukazi INSERT, UPDATE ali DELETE
  - sprožimo jih z metodo `executeUpdate`

Podatkovni strežnik in podatkovna zbirka

Podatkovni strežnik

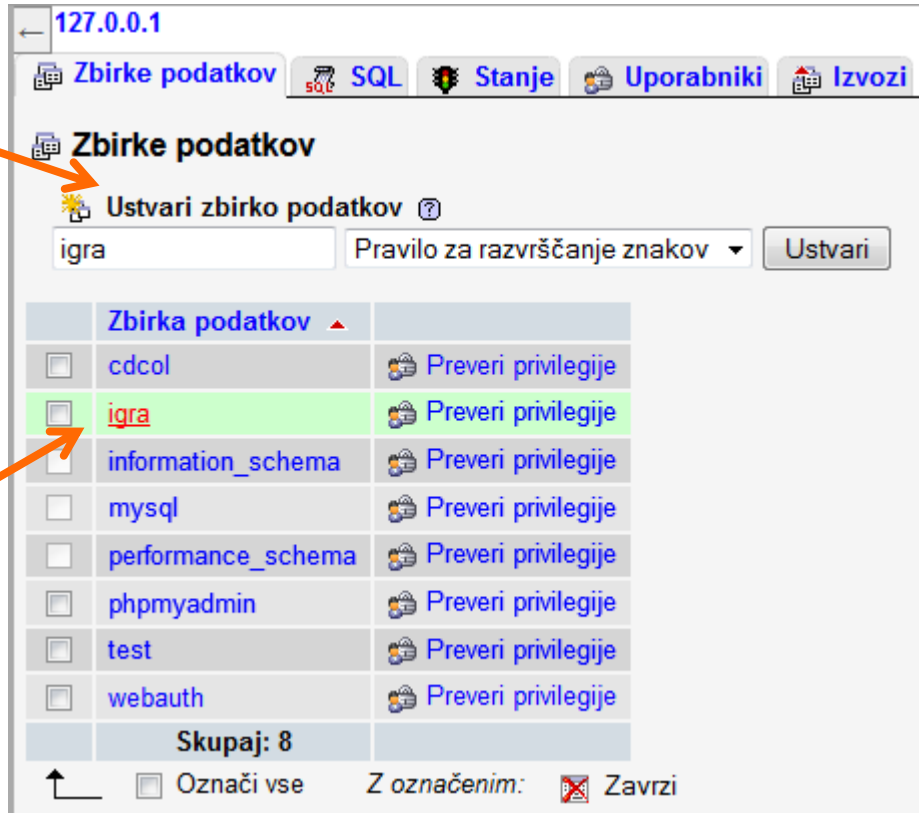
- Uporabili bomo podatkovni strežnik MySQL, ki je del paketa XAMPP
- (*) S spletnega mesta http://www.apachefriends.org na svoj računalnik prenesite zadnjo različico kompleta XAMPP za Windows
- (*) Paket XAMPP namestite v mapo **C:/xampp**
- (*) Ker ju bomo potrebovali na prihodnjih vajah, v okviru namestitve paketa XAMPP namestite tudi spletna strežnika Apache in Tomcat
- Zaženite nadzorno ploščo XAMPP-a, v njej pa še spletni strežnik Apache in podatkovni strežnik MySQL
- Poženite administratorski vmesnik podatkovnega strežnika MySQL



* Če je XAMPP že nameščen, potem je ta korak odveč

Izdelava podatkovne zbirke

- V zavihku *Databases* ustvarite novo podatkovno zbirko in jo poimenujte “igra”.
- Izberite pravkar ustvarjeno zbirko na seznamu zbirk



Podatkovni model

Podatkovni model

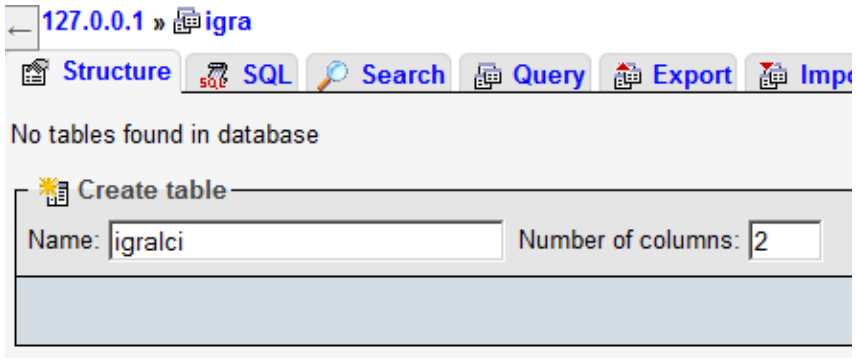
- Podatkovni model zgradimo na podlagi zahtev "naročnika" aplikacije
  - Radi bi izdelali informacijski sistem za vodenje rezultatov mobilne igre.
  - Shranjevati je potrebno rezultate vsake igre, ki jo nek igralec odigra.
  - Rezultat igre je ovrednoten s številom točk, ki jih igralec v igri doseže.
  - Igra ima tri težavnostne stopnje
    - Ker so rezultati odvisni od težavnosti igre, mora biti ob rezultatu navedena tudi težavnost
    - Težavnost bomo označili s pozitivnim celim številom (1,2,3)
  - Vsak igralec ima svoj vzdevek (oz. uporabniško ime) in geslo za prijavo v igro.

Realizacija podatkovnega modela

- V relacijskih podatkovnih zbirkah realiziramo podatkovni model s pomočjo *tabel*.
- Podatki v tabelah so zbrani v *stolpcih* (ang. fields) in *vrsticah* (ang. rows).
- Tabela ima natančno določeno število stolpcev in poljubno število vrstic.
- Stolpci predstavljajo lastnosti (atribute)
- Stolpci imajo določen podatkovni tip in omejen doseg (lahko pa tudi nabor) vrednosti
- Vrstice v tabeli predstavljajo posamezne zapise, pri čemer je zapis sestavljen iz vrednosti lastnosti (atributa), ki so definirane v stolpcih
- Potrebovali bomo dve tabeli s pripadajočimi stolpci:
  - tabela “igralci”
    - stolpca: **vzdevek**, geslo
  - tabela “rezultati”
    - stolpci: **IDigre**, vzdevek, tezavnost, rezultat

Izdelava tabel

- Ustvarite tabeli “igralci” in “rezultati“ z ustreznim številom stolpcev
- V naslednjem koraku stolpce poimenujte v skladu s podatkovnim modelom in jim določite tip in dolžino
- Izberite Storage Engine: InnoDB



Name	Type	Length/Values
vzdevek	VARCHAR	20
geslo	VARCHAR	20

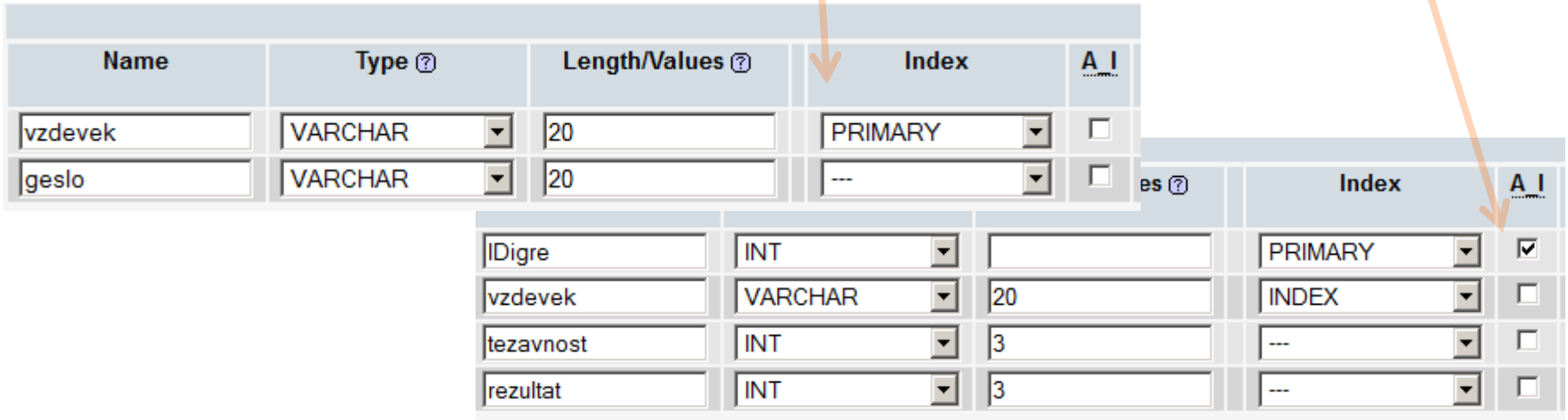
Name	Type	Length/Values
IDigre	INT	
vzdevek	VARCHAR	20
tezavnost	INT	3
rezultat	INT	3

Enoličnost (primarni ključi)

- Vsebina v različnih vrsticah tabele je lahko enaka, kar pomeni, da se lahko zgodi, da posameznih zapisov ne moremo ločiti med sabo
- Primer
  - Zagotoviti moramo enolično identiteto igralcev (preprečiti podvojene vzdevke!)
  - Ločiti moramo med posameznimi odigranimi igrami istega igralca, ki so se končale z istim rezultatom
- Podatke med seboj ločimo s pomočjo posebnega atributa, ki mu pravimo *enolični identifikator* (id)
  - Vrednosti enoličnega identifikatorja se med seboj razlikujejo, zato ta stolpec enolično razločuje med posameznimi zapisi (vrsticami) v tabeli
- Enoličnemu identifikatorju pravimo tudi primarni ključ (ang. primary key)

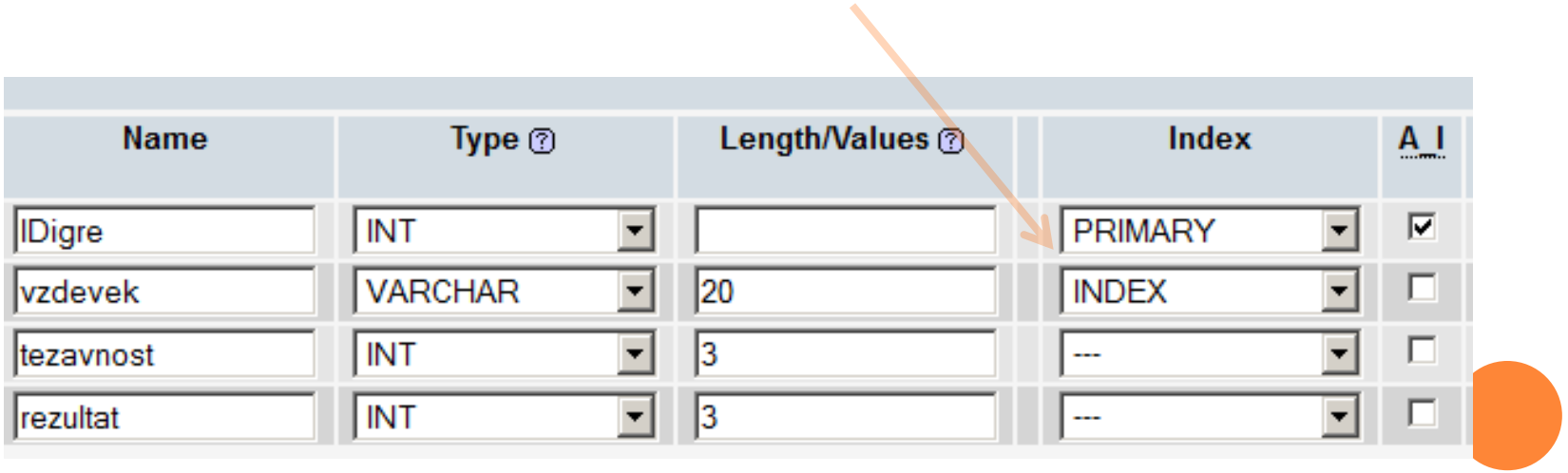
Enolični identifikator

- V našem primeru moramo enolično ločiti med posameznimi igralci in rezultati
- V MySQL nastavimo stolpec za id, tako da nastavimo želenemu stolpcu lastnost “Index“ na vrednost “PRIMARY“
- V tabeli “igralci” posebnega stolpca za id ne potrebujemo, saj lahko kot primarni ključ služi kar vzdevek
  - Vendar to pomeni, da igralec ne more več spremeniti svojega vzdevka!
- V tabeli “rezultati” je primarni ključ poseben stolpec “ IDigre”
- Če je id celo število, lahko podatkovni strežnik za vpisane vrednosti poskrbi sam
  - Ob vsakem novem zapisu vpiše vrednost, ki je za 1 večja od vrednosti predhodnega zapisa
  - V tabeli “rezultati” pri “IDigre” izberite možnost A_I (AUTO INCREMENT)



Povezave med tabelami (tuji ključi)

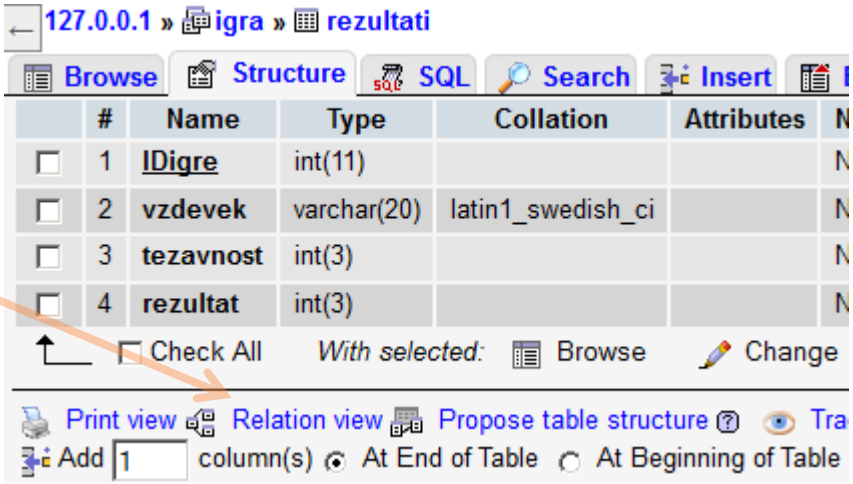
- Rezultat mora pripadati določenemu igralcu, ki pa je zapisan v drugi tabeli
  - Tabeli nista neodvisni, ampak sta medsebojno povezani preko stolpca "vzdevek"
- "vzdevek" je primarni ključ v tabeli "igralci" in *tuji ključ* (ang. foreign key) v tabeli "rezultati"
- Pri ustvarjanju tabele "rezultati" moramo stolpcu "vzdevek" nastaviti lastnost "Index" na vrednost "INDEX"

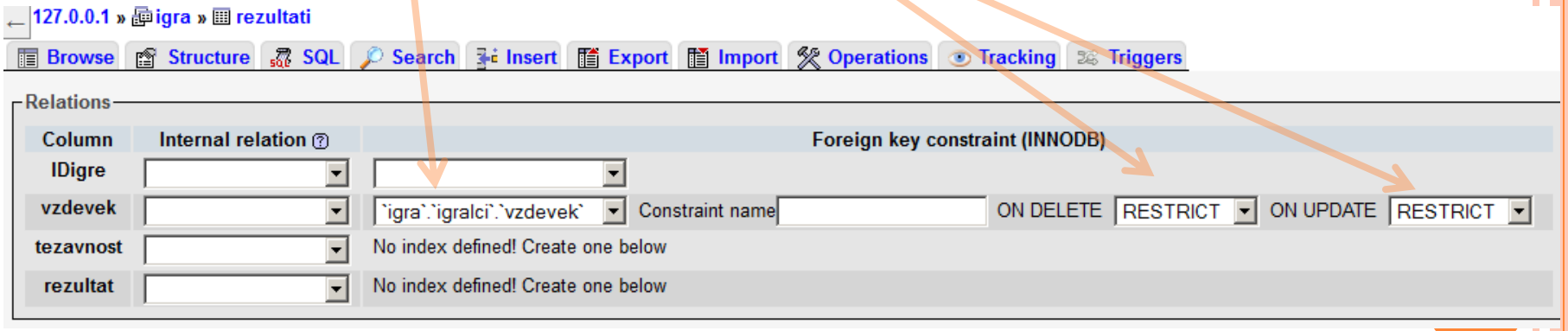


Name	Type	Length/Values	Index	A_I
IDigre	INT		PRIMARY	☑
vzdevek	VARCHAR	20	INDEX	☐
tezavnost	INT	3	---	☐
rezultat	INT	3	---	☐

POVEZAVE MED TABELAMI

- Izberemo tabelo “rezultati”, zavihek “Structure” in povezavo “Relation view”
- Stolpec “vzdevek” povežemo z istoimenskim stolpcem iz tabele “igralci” in izberemo možnost “RESTRICT”, ki zagotavlja strukturno integriteto





PROGRAMSKI DOSTOP DO PODATKOV

Priprava gonilnika

- Pripravite Eclipse
  - Ustvarite nov projekt in vanj uvozite pripravljeno kodo
- S spletnega mesta prenesite gonilnik mysql-connector-java (platform independent, zip)
  - http://dev.mysql.com/downloads/connector/j/
- V Eclipsovem projektu postavite datoteko *mysql-connector-java-xxx-bin.jar* na "Build path"
  - Desni klik na projekt → Java Build Path → Libraries → Add External JARs
- Uporabite pripravljeno kodo MySQL.java

Programski dostop do strežnika MySQL

```
private void povezi() throws SQLException,
   ClassNotFoundException {

   Class.forName("com.mysql.jdbc.Driver");

   povezava= DriverManager.getConnection(
         "jdbc:mysql://localhost/igra?” + "user=root&password=");

    poizvedba = povezava.createStatement();
}
```

Naloži gonilnik za MySQL

Ustvari povezavo do zbirke

Ustvari objekt, ki služi za pošiljanje SQL poizvedb podatkovni zbirki

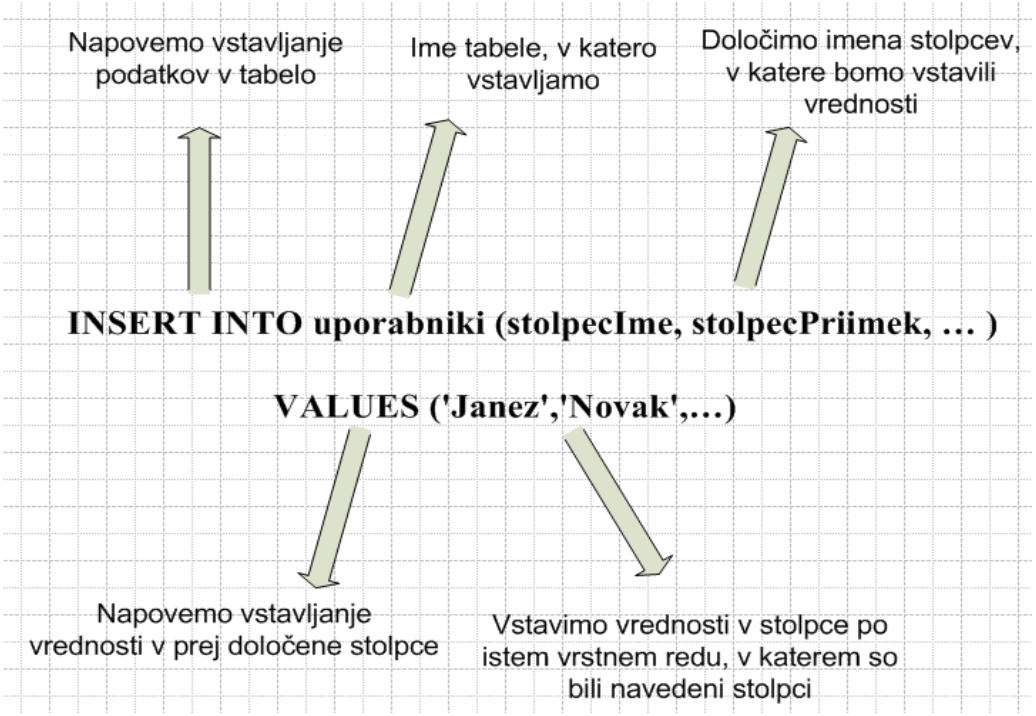
V praksi zaradi varnosti nikoli ne uporabite uporabnika “root” s praznim geslom, temveč ustvarite novega uporabnika, mu dodelite geslo in pravice, ki jih potrebuje in ga uporabite v povezavi

Sprostitev virov

- Ko odrabimo povezavo ter objekta za izvedbo poizvedb in shranjevanje njihovih rezultatov, zasežene vire sprostimo

```
private void sprosti() {
   if (povezava != null) {
         povezava.close();
   }
   // enako storimo še za ostala dva objekta
}
```

Vpisovanje v podatkovno zbirko



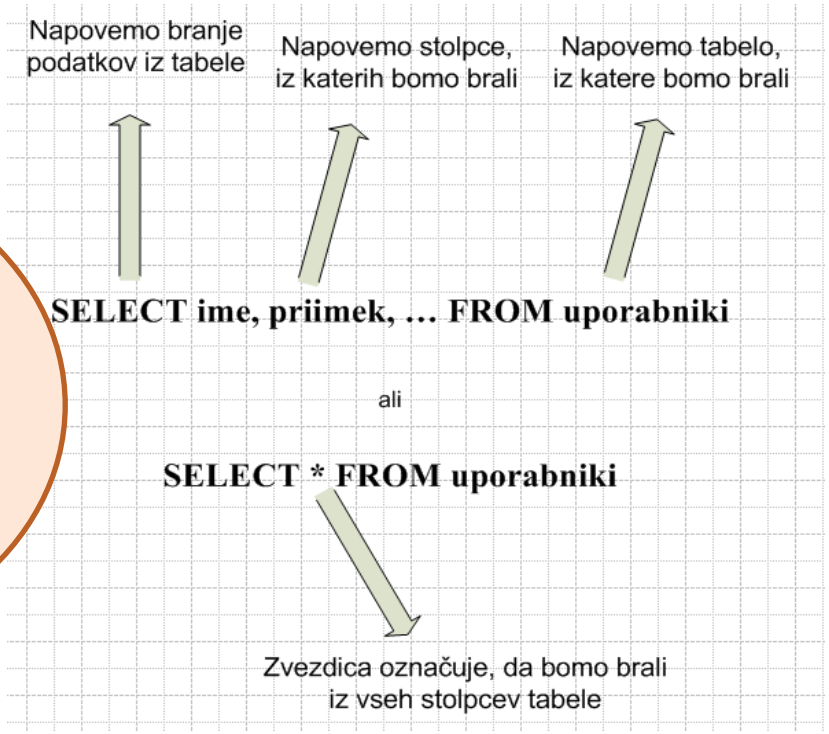

```
poizvedba.executeUpdate("INSERT INTO igra.rezultati (vzdevek, tezavnost,
   rezultat) VALUES ('"+ vzdevek + "'," + tezavnost + "," + tocke + ")");
```

Znakovne nize v SQL stavku zapisujemo v enojnih narekovajih!

Branje iz podatkovne zbirke

Izvršba poizvedbe. Rezultat poizvedbe se shrani v objekt tipa ResultSet

Do podatkov v stolpcih lahko dostopamo preko imena stolpca ali preko zaporedne številke stolpca (štetje se začne z 1 !!!) npr. resultSet.getString(1); resultSet.getString(2);




```
rezultat = statement.executeQuery("SELECT * FROM igra.igralci");

while(rezultat.next()) {
   String vzdevek = rezultat.getString("vzdevek");
   String geslo = rezultat.getString("geslo");
}
```

Programska obravnava rezultatov

- Iz metode, ki bere podatke iz zbirke, bi radi podatke vrnili z uporabo "klasičnih" javanskih struktur (npr. polj)
- Po drugi strani pa se navadno poljem skušamo izogniti, ker
  - je količina pridobljenih podatkov iz zbirke vnaprej nepoznana
  - manipuliranje z elementi polj je težavno
- Namesto polj zato raje uporabimo eno izmed podatkovnih struktur - "zbirk" (ang. collection), npr. "prilagodljivo polje" ArrayList.
- Podatkovno strukturo ArrayList lahko kombiniramo z generično kodo
- Generična koda omogoča varno uporabo podatkovnih tipov v javanskih strukturah ob času prevajanja ter odpravlja potrebo po pretvarjanju tipov

```
List<String[]> rezultatList = new ArrayList<String[]>();
while(rezultat.next()) {
    rezultatList.add(new String[]
        {rezultat.getString(1), rezultat.getString(2)});
}
```

Naloga

- Dokončajte razred MySQL
  - Kaj mora razred omogočati, določa vmesnik IgraStorable.java, zato naj razred izdela ta vmesnik
- Izdelan razred MySQL uporabite v preprostem programu
  - izdelajte nov razred z metodo `main`
  - v metodi `main` izdelajte novo različico razreda MySQL in preizkusite delovanje njegovega javnega vmesnika

IgraStorable.java

Naloga: Izdelava Javadoc dokumentacije

- Izdelajte Javadoc dokumentacijo razreda MySQL, tako da ga bodo lahko uporabljali razvijalci igre na različnih terminalih
  - Postavite kazalec v neko metodo in pritisnite ALT + Shift + J
  - Eclipse bo ustvaril Javadoc komentarje, s pomočjo katerih lahko razred dokumentirate
  - Izpolnite komentarje
  - Z izbiro Project → Generate Javadoc … ustvarite Javadoc dokumentacijo

Literatura

- Mesojedec, Fabjan: Java2, Temelji programiranja
- Preiščite splet, npr. http://stackoverflow.com/questions/11771440/creating-a-web-service-in-java-eclipse-to-get-values-from-a-database
- http://www.vogella.com/articles/MySQLJava/article.html

Spletna aplikacija

Grega Jakus

12-11-2013

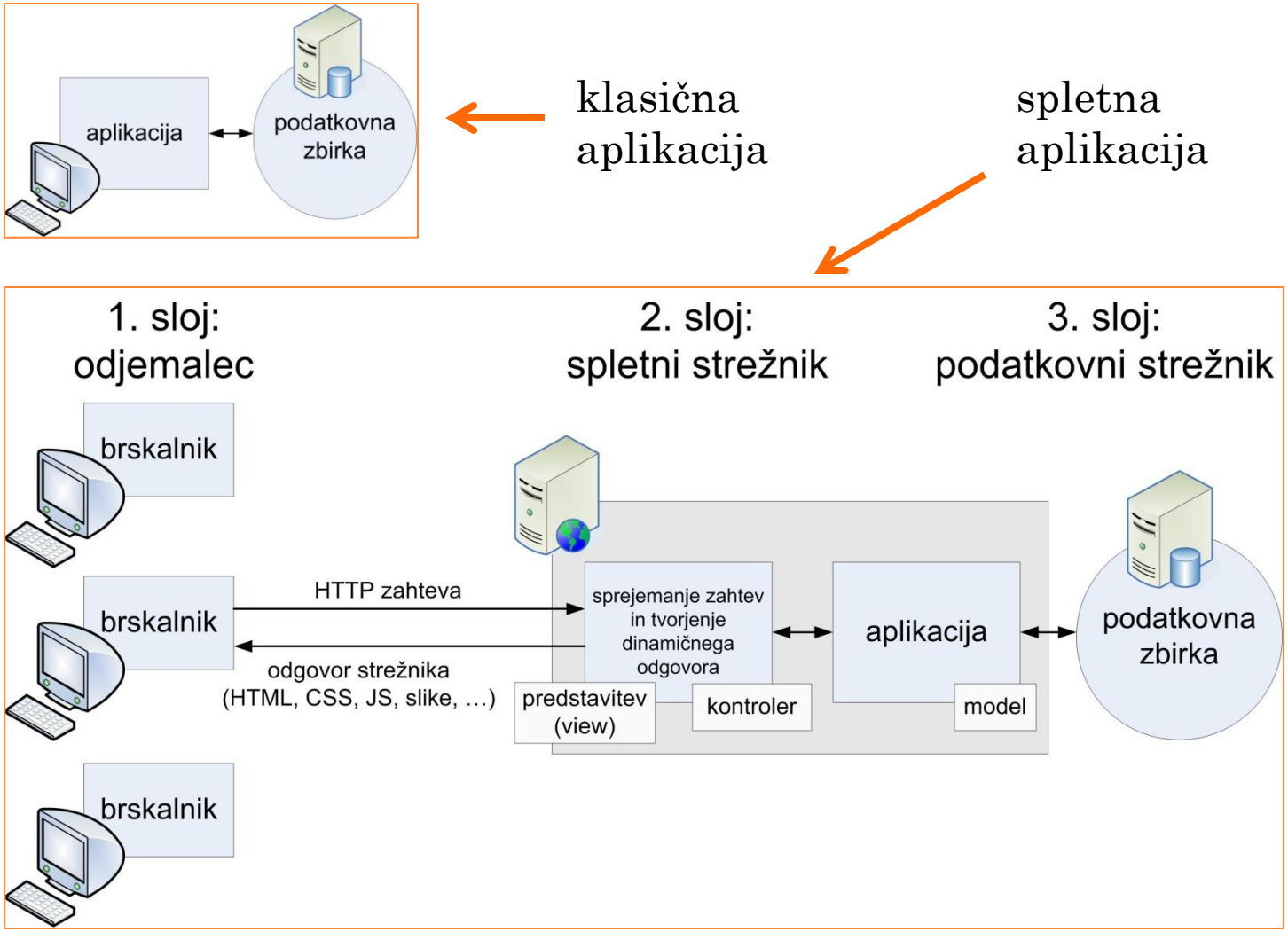
Do danes

- Naučili smo se osnov objektnega programiranja
- Izdelali smo preprosto objektno naravnano aplikacijo in njeno uporabo ponudili drugim
  - a uporabljamo jo lahko le iz drugih javanskih programov
  - vsak, ki želi uporabljati aplikacijo, mora
    - na podlagi izpostavljenega API narediti svoj javanski program
    - ali pa uporabiti enega izmed obstoječih programov (distribucija programa?)
- Kako zagotoviti večjo dostopnost naše aplikacije?
  - po možnosti tudi uporabnikom, ki niso vešči programiranja
  - aplikaciji dodati grafični uporabniški vmesnik
  - omogočiti tudi dostop uporabnikom in napravam, ki ne gostijo javanske platforme
- Ena izmed možnosti so *spletne aplikacije*

Spletna aplikacija

- Aplikacija
  - uporabniška programska oprema
  - nasprotje sistemske programske opreme
- Spletna aplikacija je aplikacija, ki je dostopna preko spleta
  - strežnik, ki nudi aplikaciji okolje za njeno delovanje, imenujemo *aplikacijski strežnik*
- Prednost spletnih aplikacij
  - Uporabnik s spletno aplikacijo komunicira preko brskalnika, brskalnik pa najdemo na praktično vsakem računalniku
  - Ni potrebe po distribuciji dodatne programske opreme na računalnike, ki želijo aplikacijo uporabljati
- Odjemalec je lahko vsak brskalnik, ki razume jezike HTML, JS, CSS in XML

Arhitektura spletne aplikacije



Vzorec Model-Predstavitev-Krmilnik

- Najbolj razširjen arhitekturni vzorec spletnih aplikacij je vzorec MVC – Model, View, Controller
- "Model" je srce aplikacije
  - Vključuje aplikacijske podatke, programsko logiko, poslovna pravila itd
  - Naš model je razred ZbirkaIgra
- Element "View" (predstavitev) je zadolžen za predstavitev podatkov, stanja aplikacije itd
- Krmilnik je posrednik med modelom in predstavitvijo
  - Sprejema ukaze od odjemalca in jih uveljavlja na pogledu in modelu
- Prednosti vzorca MVC sta delitev nalog in ponovna uporabljivost kode

- Model že imamo, izdelati moramo še predstavitev (view) in krmilnik.

Javanske spletne aplikacije

- Javanske aplikacije lahko uporabimo kot spletne aplikacije s pomočjo tehnologije "Java Servlet"
- Java Servlet je javanska alternativa strežniškim tehnologijam, kot sta PHP in ASP
- Servlet je razred, ki je sposoben sprejemanja HTTP (ali drugačnih) zahtev in vračanja odgovora na te zahteve
- Tehnologija Java Servlet nadgrajuje klasični spletni strežnik
  - razširja njegovo funkcionalnost, tako da omogoča uporabo javanskih aplikacij za tvorjenje dinamičnega odgovora odjemalcu
  - komponente spletnih strežnikov, ki omogočajo uporabo Servletov imenujemo spletni kontejnerji (web containers)
  - interakcije med kontejnerjem in servletom določa Servlet API (paket `javax.servlet`)

Spletni strežnik

Strežnik Apache Tomcat

- Apache Tomcat je spletni strežnik s kontejnerjem
- Osnovne komponente strežnika so
  - kontejner Catalina
  - spletni strežnik Coyote
  - pogon za skripte JSP (Java Servlet Pages)
    - JSP je skriptni jezik, alternativa PHP, ASP

Namestitev strežnika Apache Tomcat*

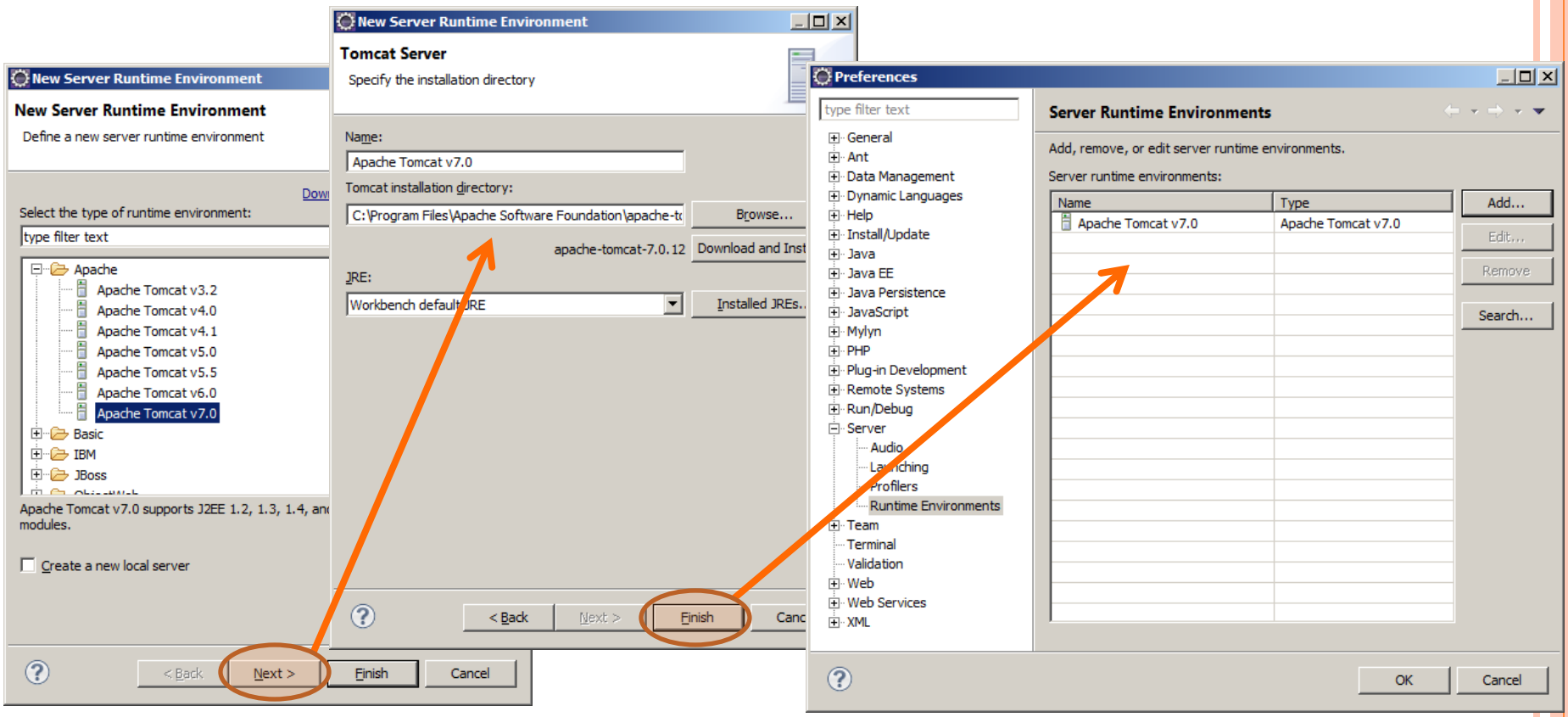
* Če je strežnik Tomcat že nameščen v okviru paketa XAMPP potem je ta korak odveč

- S spletnega mesta http://tomcat.apache.org/download-70.cgi prenesite distribucijo strežnika Apache Tomcat v obliki ZIP arhiva za vašo platformo (v učilnici so nameščeni 32-bitni Windowsi)
- Vsebino arhiva razširite v mapo **d:/seminar/tomcat/**
- Strežnik poženite z zagonom paketne datoteke **startup.bat**, ki jo najdete v mapi **/apache-tomcat-xx.xx.xx/bin/**
- Delovanje strežnika preizkusite, tako da v URL vrstico brskalnika vpišete http://localhost:8080

Integracija Apache Tomcat v Eclipse

Da bi lahko spletne aplikacije poganjali neposredno iz okolja Eclipse, je potrebno strežnik integrirati v to okolje

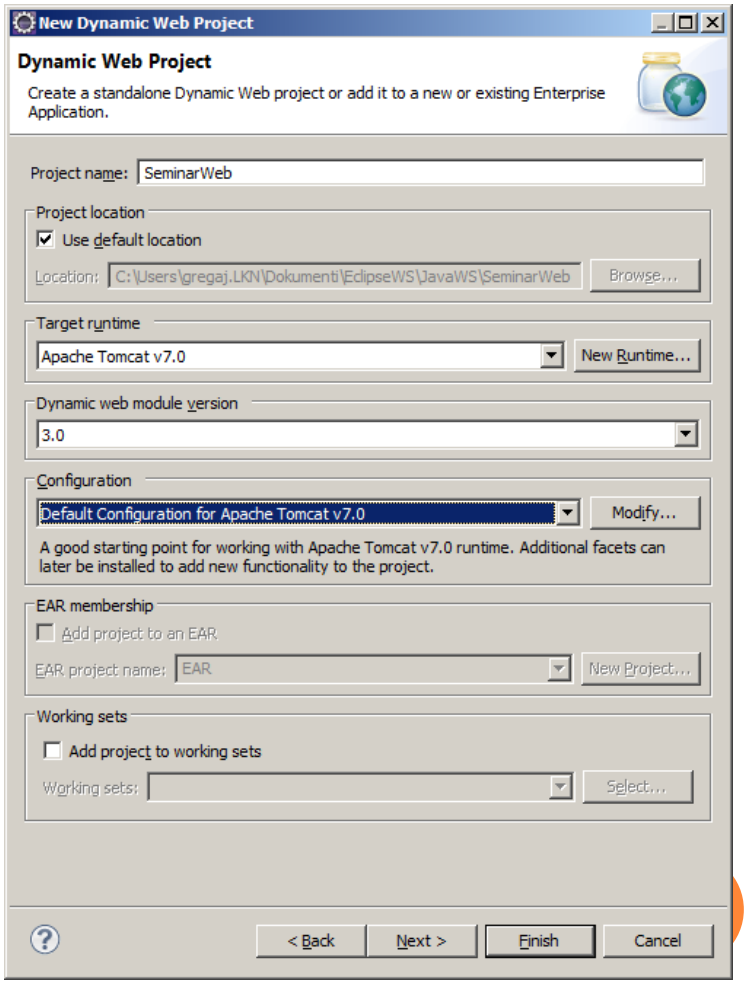
- V menijski vrstici izberite Window / Preferences / Server / Runtime Environments



SERVLET

Razvoj servletov v Eclipsu

- Servlet najlažje razvijamo in poganjamo v okviru projekta vrste Dynamic Web
- Uporabimo Java EE perspektivo
  - perspektiva je nabor aktivnih pogledov in orodij v Eclipsu
- Ustvarimo nov servlet
  - desni klik na projekt → new → Servlet
- Spletne aplikacije bomo shranjevali v paket si.uni_lj.fe.seminarTK.**web**
- Servlet poženemo z desnim klikom na servlet → Run As → Run on Server



SERVLET

```
package si.uni_lj.fe.seminarTK.web;

//importi

@WebServlet("/PozdravljenServlet")
public class PozdravljenServlet extends HttpServlet {
    public void doGet(HttpServletRequest request,
                      HttpServletResponse response)
                      throws ServletException, IOException {

   //koda servleta na naslednji strani
   }
}
```

Oznaka označuje, da je servlet, dostopen na podanem relativnem URL naslovu

Servlet, ki obravnava HTTP zahteve, mora dedovati razred HttpServlet

Parametra v metodo doGet sta objekta request (z vsebino zahteve) in response (z vsebino odgovora)

Metodo doGet pokliče kontejner samodejno, ko dobi na URL naslov servleta zahtevo vrste GET. V primeru metode POST je klicana metoda doPost

PozdravljenServlet.java

Sprejemanje HTTP zahtev in priprava odgovora

```
//sprejemanje parametrov iz zahteve
String ime = request.getParameter("ime");

//preverjanje veljavnosti parametrov
if(ime!=null && (ime = ime.trim()).length() != 0) {
  //naredi nekaj
}

// priprava odgovora brskalniku
PrintWriter odgovorBrskalniku = response.getWriter();
odgovorBrskalniku.println("Pozdravljen, " + ime+"!");
```

objekt request vsebuje vsebino zahteve

objekt response vsebuje vsebino odgovora

- V splošnem je odgovor brskalniku spletna stran v jeziku HTML

PozdravljenServlet.java

URL NASLOV SERVLETA

- URL naslov, na katerem je Servlet dostopen na spletu, je sestavljen na sledeč način

"Deployment descriptor"

- Nekateri servleti imajo v datoteki WebContent/WEB-INF/Web.xml t.i. "deployment descriptor"
- Deployment descriptor je nujen pri tistih servletih, ki ne uporabljajo oznak (annotations)

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns="http://java.sun.com/xml/ns/javaee" … >
  <display-name>Pozdravljen Servlet</display-name>
  <servlet>
     <servlet-name>Pozdravljen Servlet</servlet-name>
     <servlet-class>si.uni_lj.fe.seminarTK.web.PozdravljenServlet</servlet-class>
     <load-on-startup>1</load-on-startup>
  </servlet>
  <servlet-mapping>
    <servlet-name>Pozdravljen Servlet</servlet-name>
    <url-pattern>/servlet/*</url-pattern>
  </servlet-mapping>
</web-app>
```

Naloge

- S pomočjo brskalnika preizkusite delovanje servleta PozdravljenServlet

- Aplikacijo Igra (razred ZbirkaIgra iz prejšnje vaje) spremenite v spletno aplikacijo.
  - Razred ZbirkaIgra naj v MVC vzorcu predstavlja model
  - Servleti naj bodo krmilniki, poleg tega pa naj bodo zadolženi tudi za predstavitev (view) podatkov
    - Iz servletov bomo klicali javni vmesnik razreda ZbirkaIgra, rezultate nato ustrezno "oblikovali" ter jih posredovali brskalniku
  - Spletna aplikacija bo parametre, potrebne pri klicih javnega vmesnika razreda ZbirkaIgra, zbirala preko spleta, neposredno od končnih uporabnikov
    - za začetek s pomočjo URL vrstice brskalnika

PozdravljenServlet.java

Odjemalec spletne aplikacije

Odjemalec spletne aplikacije

- Odjemalec spletne aplikacije je spletni brskalnik
- Strežnik brskalniku posreduje spletno stran, ki služi kot grafični uporabniški vmesnik spletne aplikacije
- Spletno stran lahko izdelamo
  - znotraj servleta z uporabo razreda PrintWriter
    - pri tem gre pravzaprav za gnezdenje HTML v programski kodi
  - z uporabo jezika JSP
    - znotraj HTML dokumenta gnezdimo javansko programsko kodo
- Opazimo vzporednice z izdelavo dinamičnih spletnih strani v PHP
  - prvi pristop - gnezdenje HTML v PHP

    ```
    <?php echo "<b>Pozdravljen, "+$ime +"!</b>"?>
    ```
  - drugi pristop - gnezdenje PHP v HTML

    ```
    <b>Pozdravljen, <?php echo $ime ?>!</b>
    ```

Java Server Pages

- JSP (Java Server Pages) je tehnologija, ki omogoča dinamično tvorjenje spletnih strani na strežniku
- Princip tvorjenja strani je podoben kot v primeru PHP
  - Vsebina v HTML služi kot statična predloga,
  - v katero se vključuje dinamični del spletne strani,
    - ki je tvorjen dinamično (sproti) ob vsaki novi zahtevi po strani
- JSP dokument je torej HTML dokument, ki ima vgrajeno javansko kodo
  - Iz združene HTML + JSP kode se ustvari servlet, ki se izvede
- Delčke javanske kode, vgrajene v HTML predlogo, imenujemo "skripleti"
  - osnovni skriplet označimo z **<% %>**
  - skriplet za neposredno izpisovanje vrnjenih vrednosti označimo z **<%= %>**
  - skriplet za direktive je označen z **<%@ %>**
- V okviru Dynamic Web projekta so v Eclipsu JSP dokumenti shranjeni v mapi WebContent

Primer JSP dokumenta

```
<%@include file="Meni.html"%>

<form action="" method="post">
    Vzdevek:<input type="text" name="vzdevek" />
    …
</form>

<%@ page import="si.uni_lj.fe.seminarTK.ZbirkaIgra, java.

<% if (request.getMethod().equalsIgnoreCase("post")) {
    String vzdevek = request.getParameter("vzdevek");
            …
    if (vzdevek != null && (vzdevek = vzdevek ()).length() != 0
            ZbirkaIgra zbirka = new ZbirkaIgra ();
            zbirka.dodajRezultat(…);
%>
<p>Uspeh!</p>
```

Z direktivo “include” v spletno stran vstavimo vsebino iz druge datoteke

statična HTML koda za obrazec za vnos podatkov

uvoz razredov, na katere se bomo sklicevali v nadaljevanju

javanska koda za branje podatkov, poslanih preko metode POST

preverjanje veljavnosti podatkov

shranjevanje vpisanih podatkov z uporabo različice razreda ZbirkaIgra

Obveščanje uporabnika

dodajRezultat.jsp

Naloga

- Z uporabo jezika JSP izdelajte spletni vmesnik za dostop do spletne aplikacije Igra
- Preučite delovanje strani dodajRezultat.jsp in si z njo pomagajte pri izdelavi ostalih strani
- Vmesnik naj vsebuje
  - Izhodiščno stran
  - Stran za dodajanje igralca
  - Stran za dodajanje rezultata
  - Stran s statistiko igralca
  - Stran z lestvico najboljših igralcev
- Prehajanje med stranmi naj bo omogočeno preko preprostega menija, ki naj se nahaja na vsaki strani (glejte Meni.html)
  - Meni izdelajte enkrat, nato ga z direktivo "include" uvozite na primerno mesto na vsaki strani
- Za pošiljanje podatkov na strežnik uporabite HTML obrazec in HTTP metodo POST (glejte dodajRezultat.jsp)

dodajRezultat.jsp, statistikaIgralca.jsp, Meni.html

Spletna storitev

Grega Jakus

18-11-2013

Do danes

- Izdelali smo preprosto spletno aplikacijo in na ta način omogočili dostop do naše aplikacije širši skupini uporabnikov
  - aplikaciji smo dodali grafični uporabniški vmesnik
  - za uporabo aplikacije potrebujemo le spletni brskalnik
  - omogočili smo dostop tudi uporabnikom in napravam, ki ne gostijo javanske platforme
- Vendar…
  - Spletne aplikacije so namenjene predvsem ljudem, saj omogočajo interakcijo preko spletnih strani
  - Kaj pa programski dostop z raznolikih naprav in platform (iPhone, Android, Windows ,…) ?
- Rešitev: spletno aplikacijo bomo nadgradili/spremenili v *spletno storitev*

Klasične spletne storitve

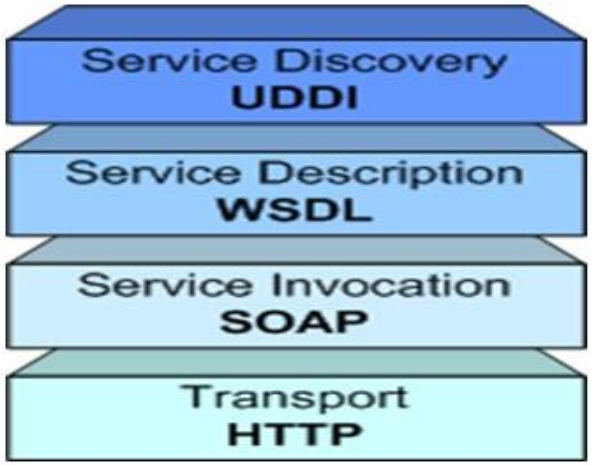
- Aplikacijske komponente,
  ki komunicirajo po odprtih protokolih
- Samostojne in samoopisljive, lahko jih "odkrivamo"
- Lahko jih uporabimo v spletnih aplikacijah
- Osnova spletnih storitev sta jezik XML in protokol HTTP
  - Jezik XML omogoča zapisovanje in pošiljanje kompleksnih sporočil in funkcij med različnimi platformami, neodvisno od programskih jezikov, ki jih te platforme uporabljajo
  - HTTP je najbolj uporabljen internetni protokol, na njem temelji tudi splet
- Način izvedbe klasičnih spletnih storitev v javi določa standard JAX-WS

Uporaba

- Spletne storitve uporabimo predvsem kadar
  - načrtujemo omrežno interakcijo med raznolikimi avtonomnimi napravami, platformami in programskimi komponentami
  - želimo povezati obstoječe nezdružljive aplikacije na različnih platformah
  - želimo izdelati programske komponente, ki bodo ponovno uporabljive

Elementi platforme spletnih storitev

- UDDI
  (Universal Description, Discovery and Interation)
- WSDL
  (Web Services Description Language)
- SOAP
  (Simple Object Access Protocol)



- Mehanizem UDDI omogoča o**dkrivanje spletnih storitev.**
- Spletne storitve so samoopisljive. Ko je neka spletna storitev odkrita, bo **svoje delovanje** in **način uporabe opisala** s pomočjo jezika WSDL.
- Klic spletne storitve je **izmenjava sporočil** med uporabnikom spletne storitve (npr. metodo, ki spletno storitev kliče) in spletno storitvijo. Izmenjava sporočil poteka po protokolu SOAP.
- Transport sporočil med odjemalcem in spletno storitvijo poteka z uporabo protokola HTTP

Orodja in pristopi pri razvoju spletnih storitev

- Izdelavo spletnih storitev olajšajo različna orodja
- V splošnem poznamo dva pristopa izdelave spletne storitve
  - "od spodaj navzgor"
    - najprej v nekem programskem jeziku izdelamo razrede, v katerih je storitev izdelana
    - orodja na podlagi izdelanih razredov samodejno ustvarijo WSDL
    - prednost pristopa je lažji začetni razvoj, po drugi strani pa je sprotno vzdrževanje originalnih razredov težje, če so ti podvrženi pogostim spremembam
  - "od zgoraj navzdol"
    - razvijalec najprej ustvari WSDL dokument
    - orodja nato na njegovi podlagi izdelajo ogrodja razredov, ki jih razvijalec dopolni s predpisano funkcionalnostjo
    - ta pristop je težji, a je bolj odporen na morebitne nadaljnje spremembe
    - dokler se ne spremenijo formati sporočil med odjemalcem in spletno storitvijo, morebitne spremembe znotraj spletne storitve ne vplivajo na delovanje in uporabo storitve

RESTful spletne storitve

- Poleg klasičnih spletnih storitev poznamo tudi t.i. RESTful (REpresentational State Transfer) spletne storitve
- Te storitve so primernejše za uporabo pri enostavnih, "sprotnih" (ad hoc) aplikacijah
  - RESTful storitve so bolj integrirane s standardom HTTP, kot to velja za klasične spletne storitve, ki temeljijo predvsem na SOAP sporočilih
  - RESTful storitve ne uporabljajo SOAP sporočil in definicij svojih vmesnikov v obliki WSDL dokumentov
  - RESTful storitve potrebujejo le osnovne in splošno uveljavljene internetne in spletne standarde, kot so to HTTP, XML, URI in MIME
  - Potrebujejo le lahko (t.i. lightweight) infrastrukturo, preprosta orodja in so poceni za uporabo ter enostavne za osvojitev

Klasične ali REST storitve?

- RESTful storitve so bolj primerne v sledečih primerih
  - Če stanje komunikacije ni pomembno (t.i. stateless storitve)
    - Dober test je razmislek, ali lahko interakcija s storitvijo “preživi" ponovni zagon strežnika?
  - V primeru, da spletna storitev ponuja podatke, ki niso (ali pa vsaj niso neprestano) dinamično tvorjeni in jih lahko predpomnimo (t.i. caching)
  - V primeru, da se storitev in njen odjemalec zavedata konteksta in vsebine, ki se prenaša.
    - REST storitve namreč ne poznajo formalnega načina predstavitve vmesnika storitve, zato morata biti obe strani predhodno seznanjeni s formatom sporočil in načinom njihove uporabe
  - V primeru, da je pomembno učinkovito koriščenje pasovne širine
    - na primer pri uporabi mobilnih terminalov in dlančnikov
  - Ko želimo integrirati spletne storitve v obstoječa spletna mesta
    - uporaba tehnologij JAX-RS, AJAX in različnih ogrodij
- Klasične spletne storitve so bolj uporabne
  - pri izdelavi zahtevnejših poslovnih (enterprise) aplikacij, ki zagotavljajo visoko stopnjo kvalitete storitev (QoS)
  - če so pomembni kvaliteta storitev, varnost in interoperabilnost med različnimi storitvami ter med storitvami in njihovimi odjemalci

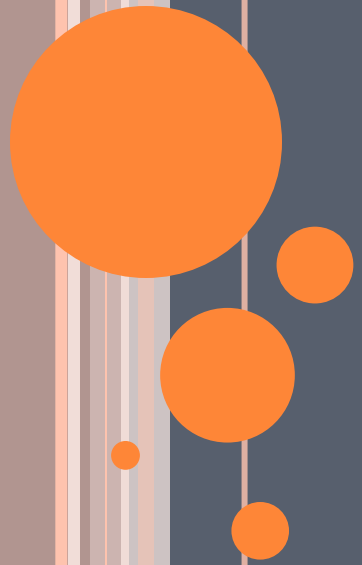
Arhitekturni vzorec REST

- RESTful storitve temeljijo na arhitekturnem vzorcu REST
- Osnovni elementi REST arhitekture so *viri* (resources)
- Strežnik zagotavlja dostop do virov, odjemalec dostopa do virov in z njimi manipulira
- Viri so identificirani z enoličnimi globalnimi oznakami (tipično URI)
- Viri so ločeni od njihove predstavitve in so tako lahko predstavljeni v različnih oblikah, npr. v XML, JSON, v obliki besedila, HTML itd.
- Do virov dostopamo z uporabo standardnih metod protokola HTTP
- Komunikacija med odjemalcem in strežnikom je omejena v smislu ohranjanja stanja komunikacije.
  - Med posameznimi zahtevami strežnik ne shranjuje stanja komunikacije s posameznim odjemalcem, npr. v obliki seje.
  - Za vsakršno ohranjanje stanja je zadolžen odjemalec. Vsaka zahteva odjemalca mora vsebovati vse informacije, ki so potrebne, da lahko strežnik uspešno servisira zahtevo.
  - Vsa izmenjana sporočila morajo zato biti samozadostna, vsakršna sprememba stanja pa mora biti v teh sporočilih izražena eksplicitno.

Metode protokola HTTP

- GET določa dostop do vira, ki nima nobenih učinkov na sam vir (branje). GET metoda zato ne more spremeniti vira.
- PUT shrani posredovan objekt (vir, entiteto) na določen URI in tako ustvari nov vir ali pa zamenja obstoječega
- DELETE odstrani želeni vir
- POST posodobi obstoječi vir ali ustvari novega

Samostalnik	Glagoli			
URL	GET	PUT	POST	DELETE
http://example.com/igra/igralci (zbirka virov)	Vrne seznam (URI) virov (igralcev) v zbirki in po možnosti ostale podatke o virih	Nadomesti celotno zbirko (igralcev) z novo zbirko	Ustvari nov vir (igralca) v zbirki. Vrne URI novega zapisa	Zbriše celotno zbirko
http://example.com/igra/igralci/igralec12 (en sam vir)	Vrne predstavitev naslovljenega vira (igralca) v izbranem MIME tipu	Nadomesti naslovljeni vir (igralca) s posredovanim novim virom. Če naslovljeni vir še ne obstaja ustvari nov vir		Izbriše vir v zbirki



Izvedba RESTful storitve

Izvedba RESTful storitve

- RESTful storitev je tipično določena z
  - osnovnim URI storitve
  - podprtimi medijskimi tipi (MIME), npr. XML, JSON, navadno besedilo itd
  - podprtim naborom operacij (POST, GET, PUT, DELETE)
- Odziv RESTful storitve je HTTP odgovor odjemalcu
- Za razliko od klasičnih storitev (temelječih na protokolu SOAP), RESTful storitve niso "uradno" specificirane, saj je REST arhitekturni vzorec, SOAP pa protokol
- Način izvedbe RESTful storitev v javi določa standard JAX-RS
  - The Java API for RESTful Web Services

JERSEY

- Jersey je t.i. referenčna izvedba (ang. “reference implementation”) specifikacij JAX-RS

- Jersey prenesemo s spletnega mesta http://jersey.java.net/download.html
  - Izberite datoteko Jersey 1.17.1 JAR bundle in jo prenesite v mapo znotraj mape d:/seminar/
  - Prenesite še knjižnico asm-3.3.1.jar (kliknite na link)
  - V Eclispu ustvarite nov (Dynamic Web) projekt, v katerem boste izdelali REST storitev
  - V zadnjem oknu pri izdelavi projekta izberite “Generate web.xml deployment descriptor”
  - Po izdelavi projekta
    - postavite obe jar knjižnici v “build path” projekta
    - ter ju poleg tega prenesite še v mapo WEB-INF/lib, kjer bosta na voljo strežniku

https://jersey.java.net/documentation/1.17/chapter_deps.html

Konfiguracija strežnika

- Jersey uporablja javanski servlet, ki skrbi za HTTP zahteve in na njihovi podlagi izbira ustrezne razrede in metode, v katerih je izdelana spletna storitev
- Jersey je potrebno registrirati kot odpravnika REST zahtev, kar storimo v konfiguracijski datoteki Web.xml
- Nadomestite vsebino konfiguracijske datoteke Web.xml z vsebino datoteke VsebinaWebXml.txt in jo prilagodite, tako da bo v skladu z vašim projektom

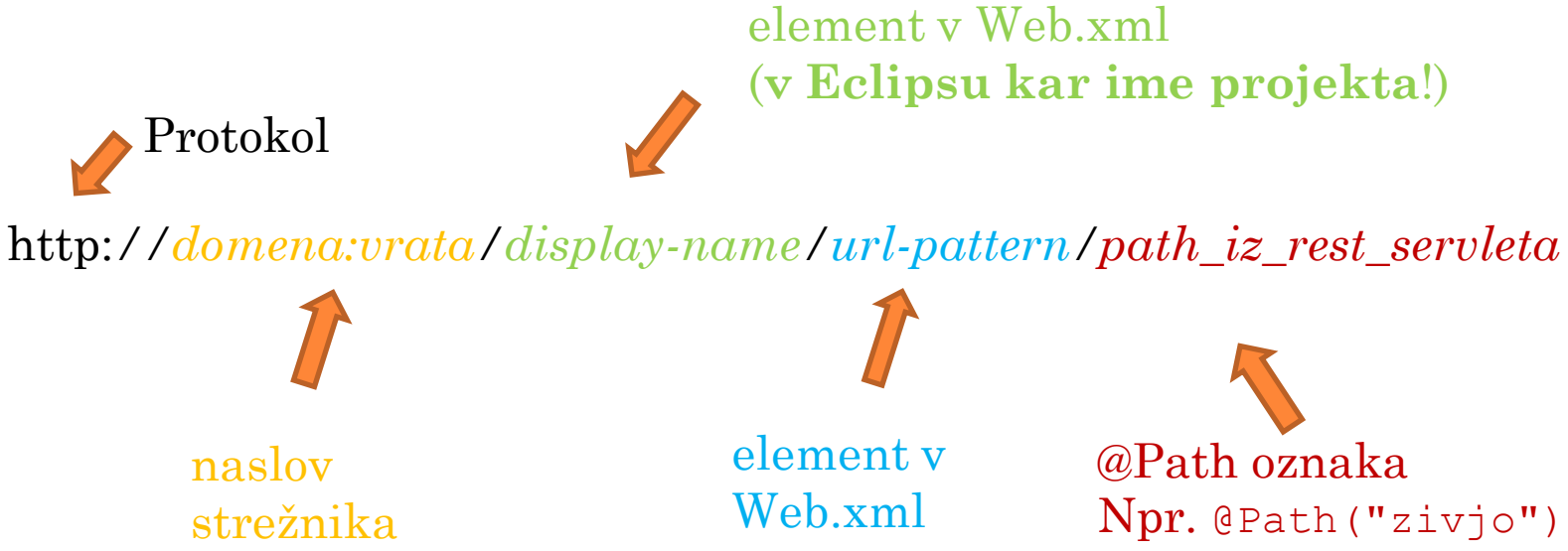
```
...
<display-name>RESTful</display-name>
 <servlet>
  <servlet-name>Jersey REST Service</servlet-name>
  <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-class>
  <init-param>
   <param-name>com.sun.jersey.config.property.packages</param-name>
   <param-value>si.uni_lj.fe.seminarTK.rest</param-value>
  </init-param>
   ...
</servlet>
 <servlet-mapping>
  <servlet-name>Jersey REST Service</servlet-name>
  <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
   ...
```

Paket, v katerem bo Jersey iskal razrede spletne storitve

VsebinaWebXml.txt

URL NASLOV STORITVE

- URL naslov, na katerem je REST storitev dostopna, je sestavljen na sledeč način



PozdravljenREST.java

Naloge

- V projekt v Eclipsu uvozite storitev PozdravljenREST.java
- Storitev poženite (→ Run on Server!)
- Storitev testirajte z uporabo brskalnika
  - Potrebno bo pravilno sestaviti URL naslov!

PozdravljenREST.java

Pretvorba javanskega razreda v storitev

- Specifikacije JAX-RS določajo posebne oznake, na podlagi katerih lahko običajen javanski razred izpostavimo kot REST storitev

Oznaka	Opis
@**PATH**(relativna-pot)	Nastavi pot do servleta na ***osnovni URL + /relativna-pot***
@**POST**, @**GET**, @**PUT**, @**DELETE**	Nakazuje, da označena metoda odgovarja na HTTP zahtevo vrste POST, GET, PUT ali DELETE
@**Produces**(tip[, drugi-tipi])	Določa, katerega MIME tipa je vsebina, ki jo vrača metoda odjemalcu storitve
@**Consumes**(tip[, drugi-tipi])	Določa, MIME tip vsebine, ki jo metoda sprejema
@**PathParam**	Uporablja se za vstavljanje vrednosti iz URL v parametre klicane metode.

Vstopna točka RESTful storitve

- Vstopna točka je **Razred korenskega vira** (Root resource class)
- Razred korenskega vira
  - je označen z oznako @Path
  - **ali** vsebuje vsaj eno metodo, označeno z oznako @Path
  - **ali** vsebuje vsaj eno **metodo virov**
- **Metode virov** so metode, ki odgovarjajo na HTTP zahteve in so označene z eno zmed oznak @GET, @PUT, @POST, ali @DELETE

Oznaka Path

- Nastavi relativno pot do storitve

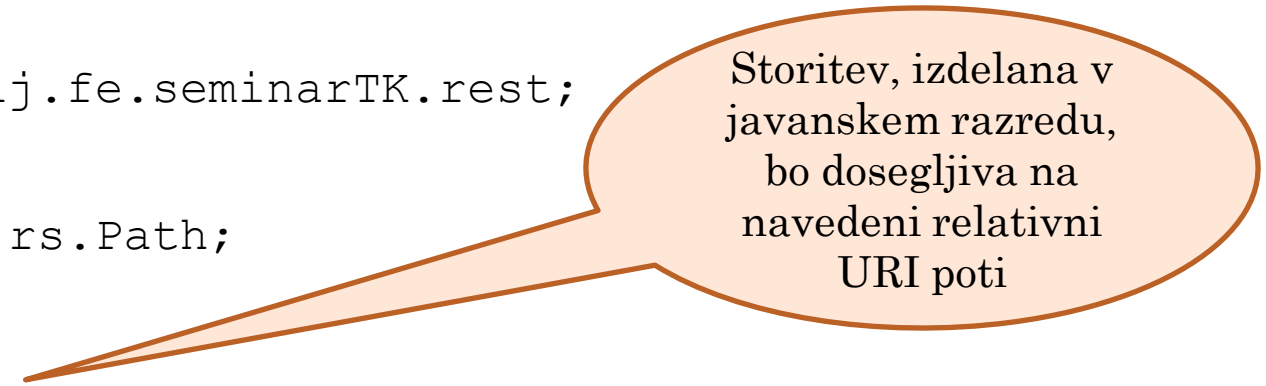
```
package si.uni_lj.fe.seminarTK.rest;

import javax.ws.rs.Path;

@Path("/zivjo")
public class PozdravljenREST{

   // metode virov in ostale metode

}
```



PozdravljenREST.java

Oznake GET, POST, Produces in Consumes

Metoda obravnava GET zahteve

Metoda bo odjemalcu poslala vsebino, določeno z navedenim medijskim tipom MIME

```
@GET
@Produces("text/plain")
public String nekaMetoda() {
   return "Pozdravljen svet!";
}
```

Metoda obravnava POST zahteve

Metoda sprejema vsebino, določeno z navedenim medijskim tipom MIME

```
@POST
@Consumes("text/plain")
public void nekaMetoda(String sporocilo){
        // koda, ki shrani sporočilo
}
```

Naloge

- V projekt uvozite razred PozdravljenREST_Odjemalec in ga poženite kot javansko aplikacijo
- S pomočjo razreda pokličite storitev PozdravljenREST in od storitve zahtevajte odgovor v različnih medijskih tipih

PozdravljenREST_Odjemalec.java

Parametri REST storitev

- REST storitev lahko pokličemo s parametri različnih vrst
  - **Query**
    - Parameter je del URL naslova (t.i. "query string")
      - http://www.example.com?**parameter1=vrednost1&parameter2=vrednost2**
  - **URI Path**
    - Parameter je del URI poti
      - http://www.example.com/**parameter/vrednost**
  - **Form**
    - Parameter je vrednost v HTML obrazcu
      - <input type="text" **name="parameter"**/>
      - **vrednost** vpiše uporabnik v prikazano vnosno polje
  - Cookie
    - Parameter je neka vrednost v piškotku
  - Header
    - Parameter je vrednost v glavi HTTP paketa
  - Matrix

Sprejemanje "query" parametrov

- Primer sprejemanja query parametrov (na podlagi GET zahteve)

  {*osnovni URL servleta*}/**?ime=lojze&starost=46**

```
@GET
public String reciZivjo(
   @QueryParam("ime") String ime,
   @QueryParam("starost") int starost)
```

Oznaka **@QueryParam** naznani, da so vrednosti parametrov, navedenih v deklaraciji metode, del "query" niza

- Delovanje oznake **@QueryParam** v zgornjem primeru je logično ekvivalentno:

```
reciZivjo("Lojze", 46);
```

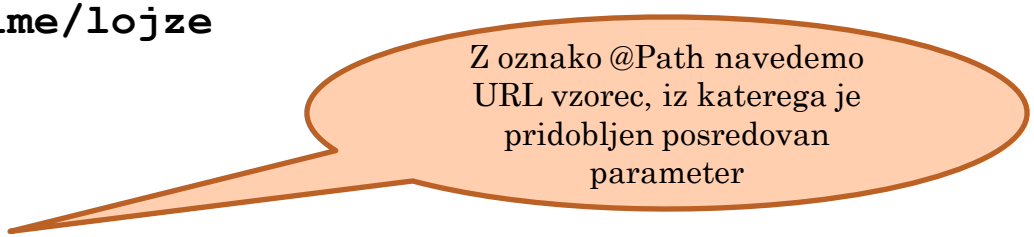
- V REST arhitekturi se izogibamo uporabi "query" parametrov za sklicevanje na vire
  - namesto teh raje uporabimo URI Path parametre
  - uporaba "query" parametrov je sprejemljiva, če so parametri opcijski (neobvezni)

REST_Parametri.java

Sprejemanje parametrov iz URL poti

- Primer sprejemanja parametrov iz URL poti (na podlagi GET zahteve):

  {*osnovni URL servleta*}/**ime/lojze**




```
@GET
@Path("/ime/{vpisanoIme}")
public String reciZivjo(@PathParam("vpisanoIme") String ime)
```

- Delovanje oznake **@PathParam** v zgornjem primeru je logično ekvivalentno

```
reciZivjo("Lojze");
```

REST_Parametri.java

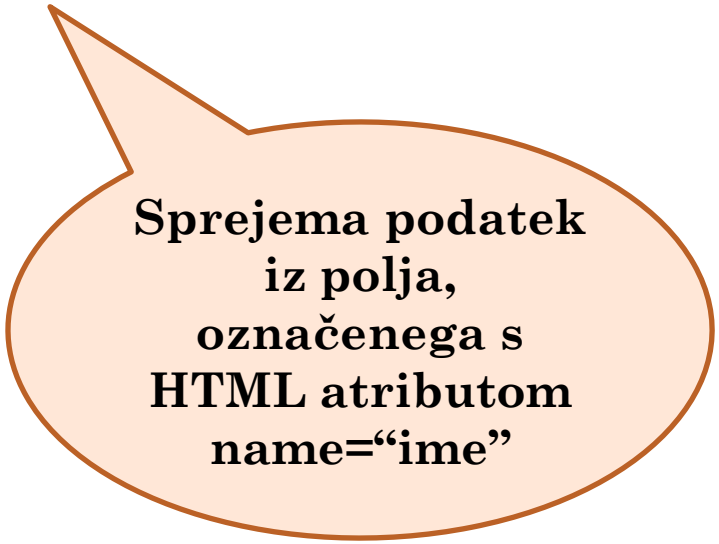
Naloga – metoda GET

- Izdelajte novo storitev z metodo, ki vrača lestvico najboljših 10 igralcev Igre na neki težavnosti.
  - Za posredovanje težavnosti uporabite "query string" parameter
- Potek
  1. Pripravite aplikacijo Igra
     - Zaženite MySQL in vanj uvozite podatkovno zbirko igra
     - Dodajte gonilnik za MySQL
       - na "build path" projekta
       - in v mapo WEB-INF/lib, kjer bo dostopna strežniku
     - V projekt s spletno storitvijo dodajte (uvozite) razred ZbirkaIgra
  2. Pripravite novo storitev
     - Nastavite ji novo relativno pot (npr. @Path("/igra"))
     - Ustvarite novo metodo storitve in jo ustrezno označite (vhodna HTTP metoda, relativna pot, izhodni medijski tip)
     - Sprejmite vhodne parametre in jih shranite v lokalne spremenljivke
     - V metodi ustvarite nov objekt ZbirkaIgra in pokličite njegov javni vmesnik (metodo, ki vrne najboljših nekaj igralcev)
     - Vrnite rezultate
  3. V brskalniku preizkusite delovanje storitve

StoritevIgra.java

Sprejemanje Form parametrov (metoda POST)

```
@POST
@Consumes("application/x-www-form-urlencoded")
public void post(@FormParam("ime") String ime){
   // shraniti sporočilo
}
```



Naloga – metoda POST

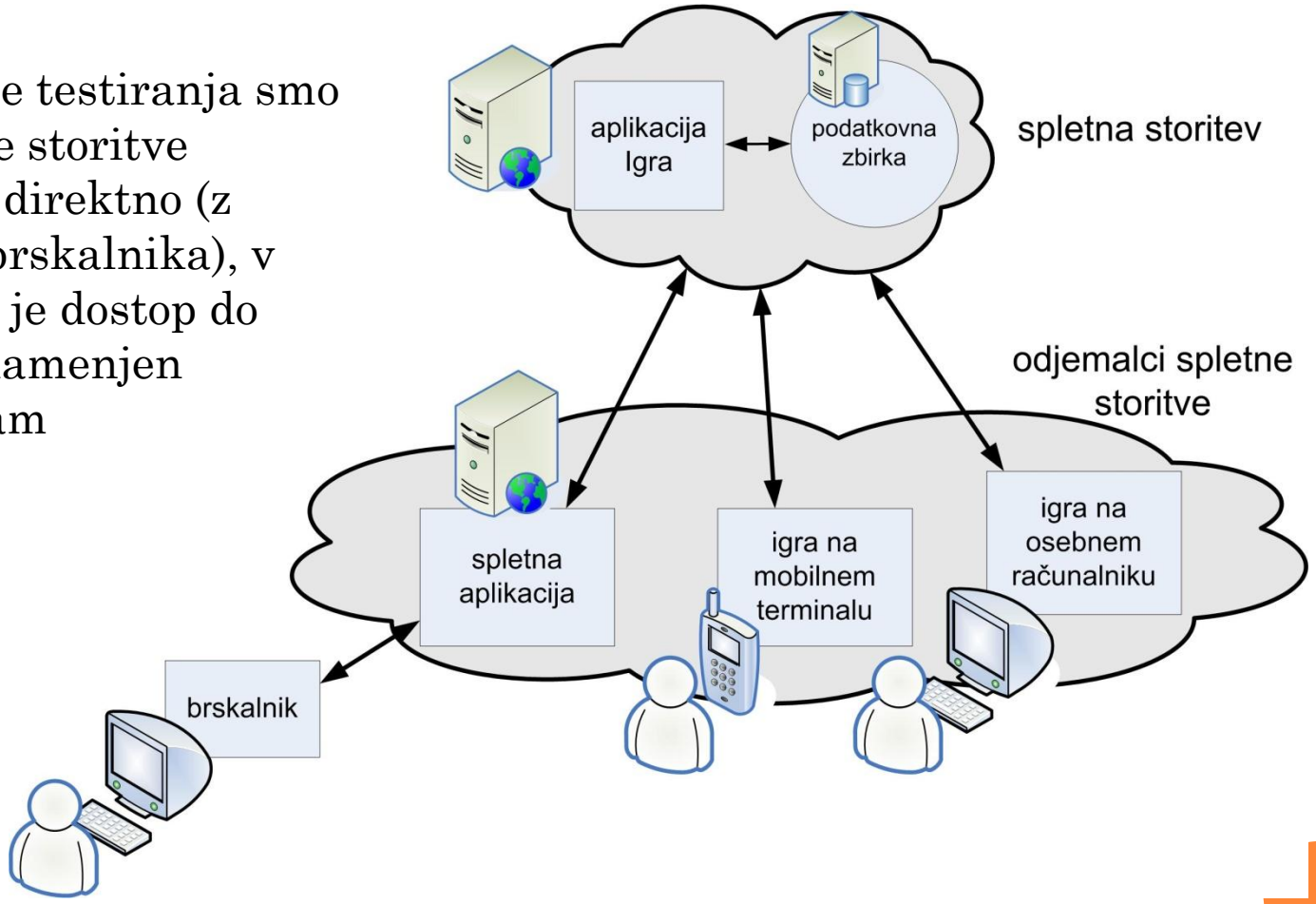
- Dopolnite izdelano storitev tako, da bo omogočala vpis novega rezultata na neki težavnosti. Uporabite metodo POST.
  1. Pripravite storitev
     - V obstoječem razredu s storitvijo ustvarite novo metodo storitve in jo ustrezno označite (vhodna HTTP metoda, relativna pot, izhodni medijski tip)
     - Sprejmite vhodne parametre in jih shranite v lokalne spremenljivke
     - V metodi ustvarite nov objekt ZbirkaIgra in pokličite njegov javni vmesnik (metodo, ki je namenjena vpisovanju novega rezultata)
     - Vrnite sporočilo o uspešnem vpisu
  2. Pripravite odjemalca storitve
     - S pomočjo obrazca v POST-test.html testirajte storitev

StoritevIgra.java, POST-test.html

Spletna aplikacija kot odjemalec spletne storitve

Arhitektura

- Za potrebe testiranja smo do spletne storitve dostopali direktno (z uporabo brskalnika), v praksi pa je dostop do storitev namenjen aplikacijam



Naloge

- Aplikacijo Igra (razred ZbirkaIgra) spremenite v spletno storitev
- Spletno aplikacijo (JSP skripte) iz prejšnje vaje spremenite v odjemalca spletne storitve
- Omogočite sledeče operacije
  - Prikaz lestvice najboljših igralcev (metoda GET)
  - Vpis novega rezultata (metoda POST)
  - Ostalo doma

- Primer (vpis novega rezultata)
  - Spletna aplikacija preko obrazca zbere podatke o vpisu novega rezultata
  - Spletna aplikacija pokliče odjemalca spletne storitve, ki podatke pošlje ustrezni metodi spletne storitve
  - Spletna storitev pokliče ustrezno metodo razreda ZbirkaIgra, ki podatke vpiše v podatkovno zbirko

Naloge - nasveti

1. Vzpostavite spletno storitev
   - Sledite poti, ki smo jo ubrali pri eni izmed prejšnjih nalog
     - Vzpostavite javansko aplikacijo Igra
     - Aplikacijo pretvorite v spletno storitev tako, da vsako metodo javnega vmesnika pretvorite v storitev
2. Izdelajte spletno aplikacijo
   - Ustvarite jo v ločenem projektu
     - ločena projekta "simbolizirata" ločena računalnika – na enem teče storitev, na drugem pa spletna aplikacija
     - vse potrebne jar knjižnice (MySQL, Jersey) postavite v "build path" projekta in jih dodatno prenesite še v mapo WEB-INF/lib, kjer bodo na voljo strežniku
   - Za izdelavo spletne aplikacije uporabite predlogi lestvica.jsp in dodajRezultat.jsp
   - V projekt (v mapo src) uvozite pripravljenega odjemalca spletne storitve (WebREST_Odjemalec) in ga pokličite iz JSP skript
     - in sicer na tistih mestih, kjer ste v spletni aplikaciji (prejšnja vaja) neposredno klicali javne metode razreda ZbirkaIgra

Namestitev ali selitev spletne storitve ali aplikacije

- Ko spletno storitev ali aplikacijo razvijemo, jo namestimo na "pravi" strežnik ("preselimo" jo iz Eclipsa)
- Spletne storitve so navadno na drugih strežnikih (računalnikih) kot njihovi odjemalci
- Selitev izdelane storitve/aplikacije je mogoča preko izvoza v obliki WAR datoteke
- Desni klik na projekt ali paket → Export… → Web → War File

- Storitev/aplikacijo selite med strežniki preprosto tako, da kopirate izvoženo WAR datoteko
  - Izvoženo datoteko postavite v mapo *webapps*, ki se nahaja znotraj namestitvene mape strežnika Tomcat, na katerega storitev selite
  - Ponovno zaženite strežnik (/bin/shutdown.bat, /bin/startup.bat)
- Pri selitvah med strežniki pazite na morebitne spremembe URL naslovov virov
  - Pazite tudi na poti do lokalnih virov, ki jih storitev naslavlja!

Literatura

- http://www.vogella.com/articles/REST/article.html#restjersey
- http://docs.oracle.com/javaee/6/tutorial/doc/giepu.html
- Knjiga “REST in practice”
- http://www.restapitutorial.com/
- Jersey: https://jersey.java.net/

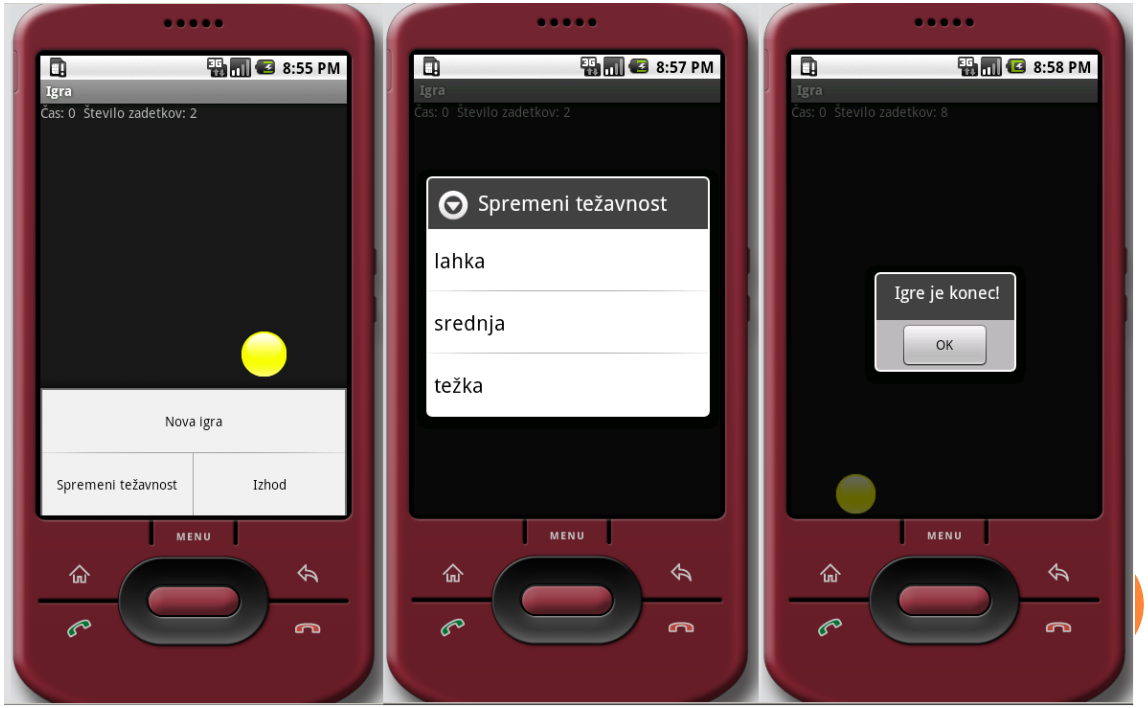
Mobilna aplikacija na platformi Android

Grega Jakus

25-11-2013

Danes

- Na vajah bomo izdelali preprosto igro, v kateri bomo uporabili nekatere funkcionalnosti platforme Android.
- Večina funkcionalnosti bo pripravljena v obliki delčkov kode, ki jih bo potrebno povezati v delujočo aplikacijo.
- Po zaslonu se z različnimi hitrostmi premika »balonček«. Naloga uporabnika je, da ga s prstom »poči«, s tem pa dobi točko.
- Igra ima omejen čas in težavnost (hitrost premikanja balončka).

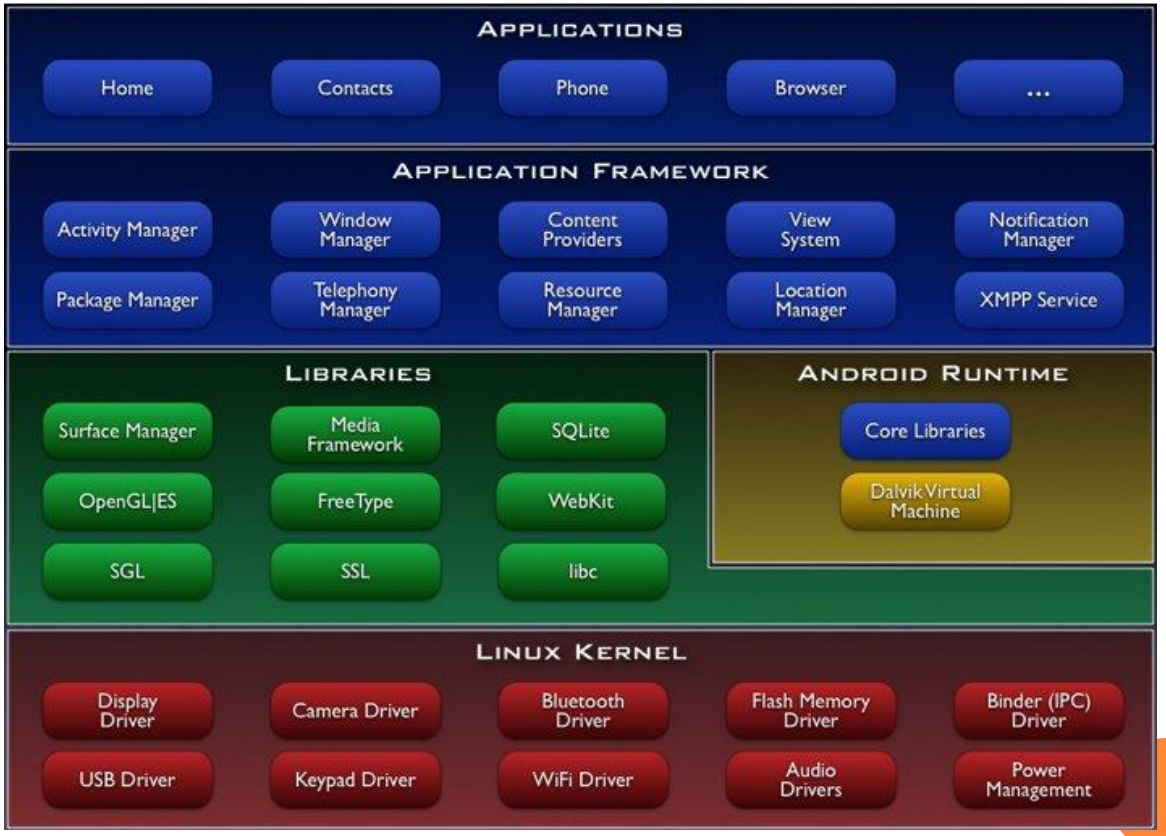


Razvoj aplikacij za mobilne naprave

- Mobilne naprave imajo nekatere specifičnosti, ki jih je potrebno upoštevati pri razvoju aplikacij.
- V primerjavi z namiznimi računalniki imajo mobilne naprave
  - manjšo moč procesiranja,
  - omejen delovni pomnilnik,
  - omejene kapacitete trajnega shranjevanja podatkov,
  - manjše zaslone z manjšo ločljivostjo,
  - manj zanesljive podatkovne povezave,
  - višje stroške prenosa podatkov,
  - počasnejši prenos podatkov z večjimi zakasnitvami,
  - omejeno trajanje baterije.

Android

- Android je operacijski sistem, zasnovan na jedru Linux
- Prilagojen za pametne mobilne telefone
- OS je zasnovalo podjetje Android Inc., sedaj pa je last podjetja Google
- Konkurenca: Symbian (Nokia), BlackBerry (RIM), iOS (Apple), Windows Mobile (Microsoft)



Razvojni kompleti in orodja

- Java SDK
  - Razvoj aplikacij poteka v programskem jeziku java in z uporabo že pripravljenih programskih knjižnic
- Android SDK
  - Programske knjižnice
    - so del aplikacijskega programskega vmesnika (API) platforme, ki predstavlja "most" med aplikacijo in platformo Android
    - API izpostavlja funkcionalnosti platforme, tako da jih lahko razvijalec enostavno uporabi s pomočjo že pripravljenih razredov, metod in ostalih elementov objektnega programiranja
  - Emulator
    - program, s katerim dosežemo posnemanje delovanja prave mobilne naprave.
    - služi za testiranje aplikacij brez uporabe mobilne naprave
  - Razhroščevalnik
  - Dokumentacija, vzorčna programska koda, tečaji, …
- Razvojno okolje (npr. Eclipse)

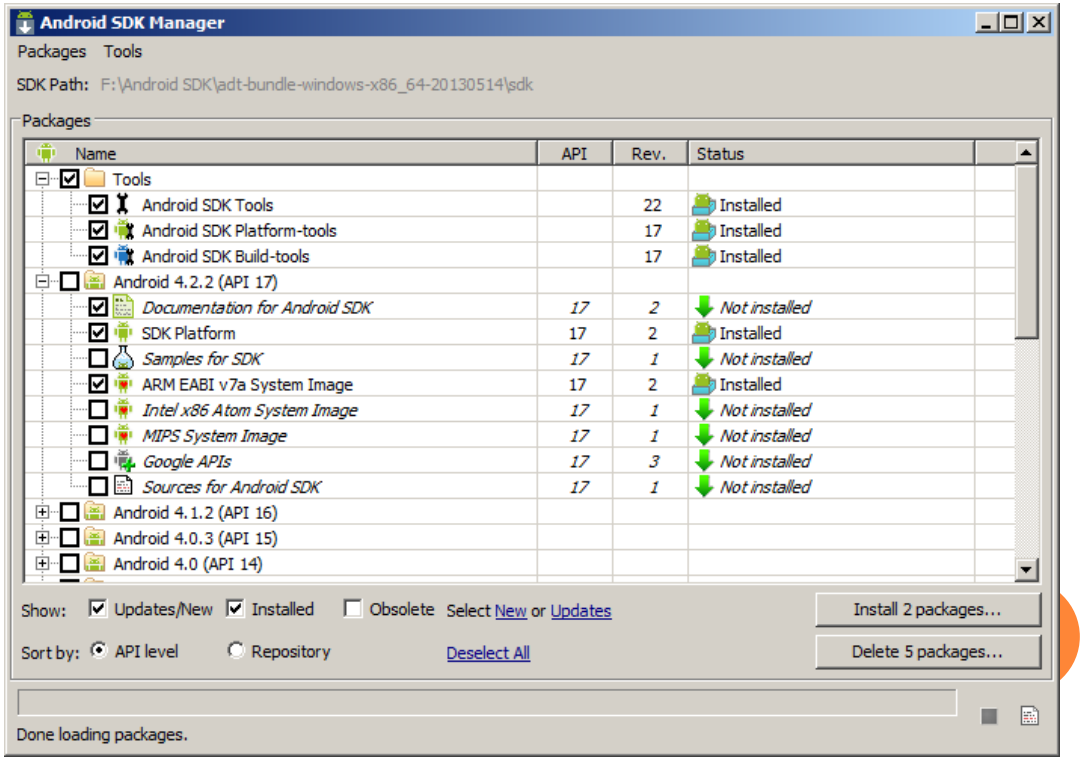
Vzpostavitev razvojnega okolja

Java in Android SDK

- S spletnega mesta http://developer.android.com/sdk prenesite na svoj računalnik Androidov SDK (»ADT bundle for Windows«).
- Vsebino *zip* datoteke razširite v mapo **d:/seminar/android/**
- Razglejte se po vsebini mape. Ta vsebuje
  - Okolje Eclipse, pripravljeno za delo na platformi Android (ADT)
  - Razvojni komplet SDK
  - Upravljalnik razvojnega kompleta SDK (SDK manager)

Namestitev komponent SDK

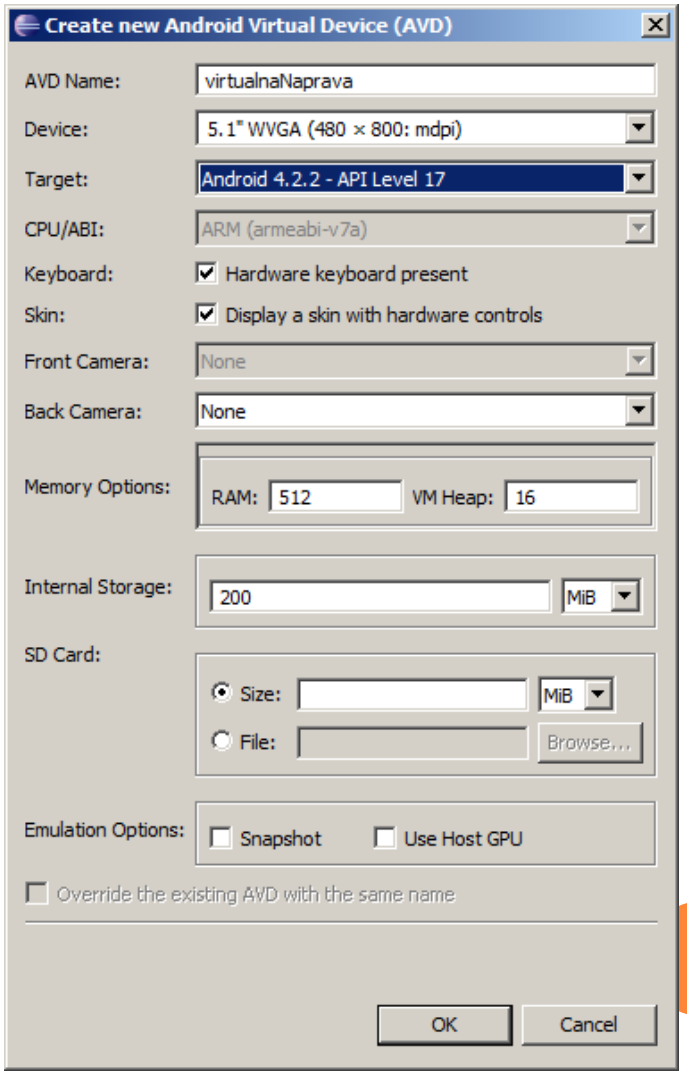
- Poženite Eclipse ADT
- Izberite pot do delovnega prostora **d:/seminar/workspace** in nastavite ta delovni prostor za privzeti delovni prostor
- Izberite Window → Android SDK Manager in izberite namestitev komponent, kot to prikazuje slika.



Nastavitev emulatorja

- Za nastavitev emulatorja za uporabo neposredno iz okolja Eclipse izberite Window → Android Virtual Device Manager

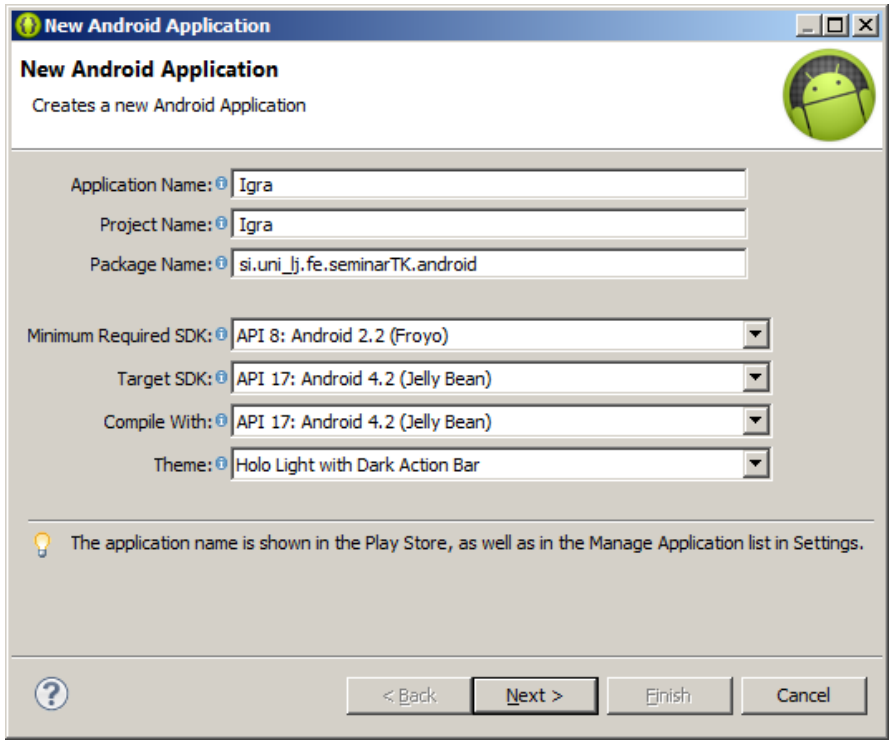
- Izberite **New,** izpolnite polja, kot to prikazuje slika, in nato izberite **OK**.



Priprava projekta in viri aplikacije

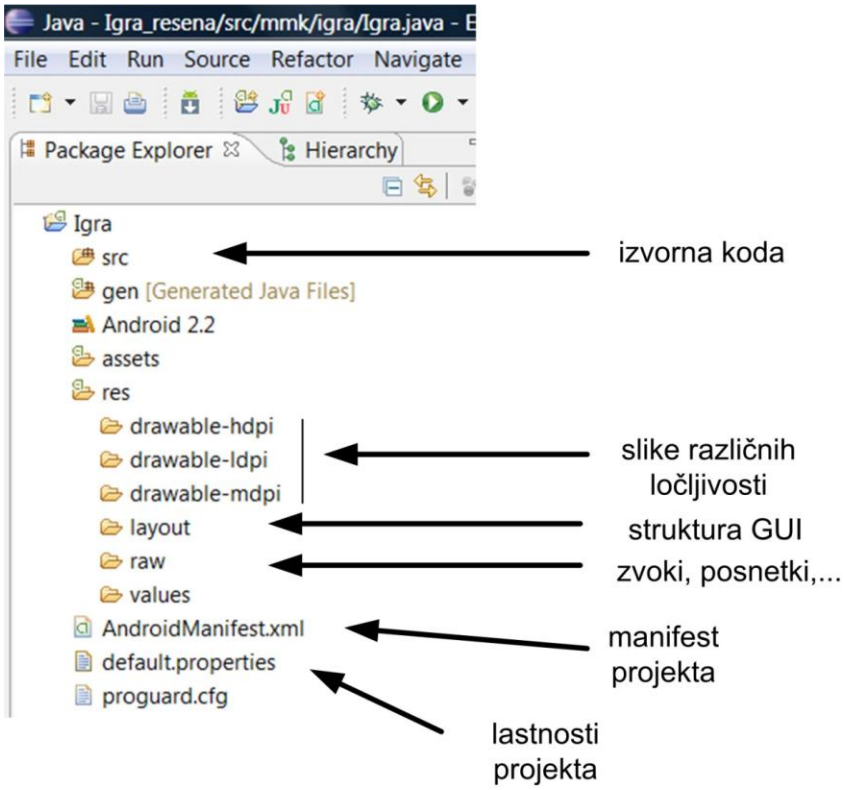
Projekt

- Ko je razvojno okolje pripravljeno za uporabo, lahko ustvarimo nov Androidov projekt
  - File → New → Project… → Android → Android Application Project
- Med razvojem lahko aplikacijo preizkušamo v emulatorju
  - klik z desnim miškinim gumbom na ime projekta → Run As → Android Application.
  - Zaganjanje emulatorja traja kar precej časa, zato ga ne izklapljajte med posameznimi testiranji aplikacije.



Ustvarite nov projekt in ga poimenujte »Igra«. Obrazec izpolnite tako, kot kaže slika.

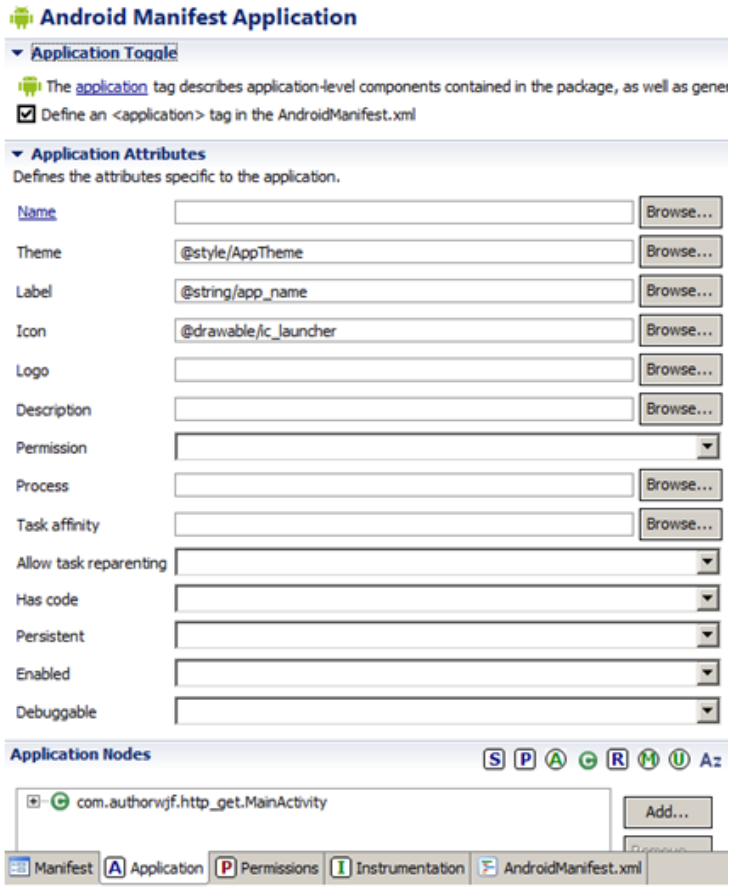
Viri, ki tvorijo aplikacijo



- Na naslovu … najdete zip arhiv s pripravljeno kodo in ostalimi viri
- V mapi *res* ustvarite mapo *raw*. V to mapo bomo shranjevali zvoke
- Prenesite slike v ustrezne mape

Manifest aplikacije

- Manifest je XML datoteka v izhodiščni mapi projekta
- Androidovi platformi predstavlja najpomembnejše informacije o aplikaciji, ki so potrebne pred samim izvajanjem programske kode
  - komponente aplikacije, njihove zmožnosti in potrebe, potrebna dovoljenja itd
- Več na http://developer.android.com/guide/topics/manifest/manifest-intro.html

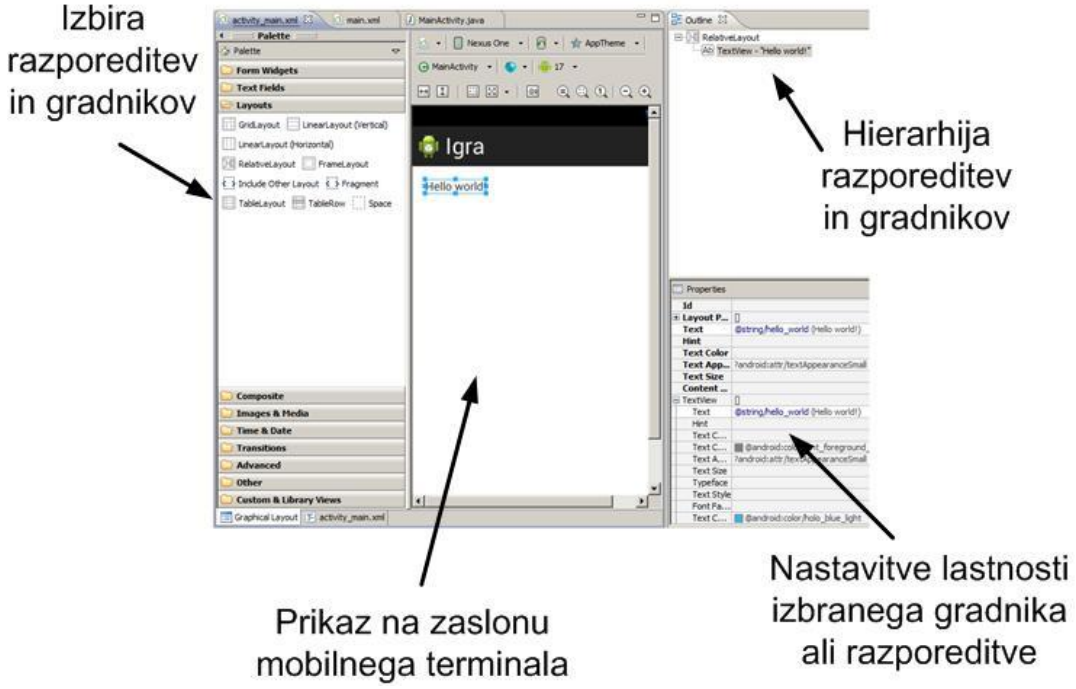


Odprite manifest aplikacije. Kot ikono aplikacije nastavite sliko balončka.

Grafični uporabniški vmesnik

Grafični uporabniški vmesnik

- Struktura grafičnega uporabniškega vmesnika je definirana v XML datoteki v mapi **res/layout**
- Vizualna vsebina zaslona aplikacije je strukturirana v hierarhiji *pogledov* (*views*) ter ostalih gradnikov.
- Način razporejanja gradnikov znotraj starševskega gradnika dosežemo z *razporeditvami* (*layouts*)

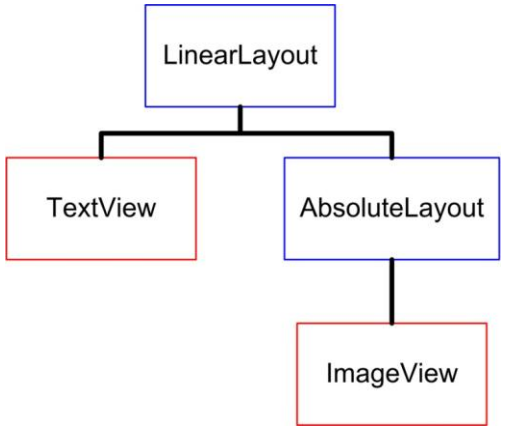


Naloge

- V naši aplikaciji bo vmesnik uporabljal razporeditev *LinearLayout*, v katero bomo umestili pogled *TextView* in razporeditev *RelativeLayout*. V slednjo bomo dali še pogled *ImageView*

Razporeditve in poglede morate poimenovati natančno tako, kot so poimenovani na sliki, ker so ta imena uporabljena v pripravljeni kodi!

- V zavihku *Properties* gradnikom nastavite lastnosti v tabeli
- Ko definirate strukturo grafičnega vmesnika, lahko zametke aplikacije že preizkusite v emulatorju



gradnik	lastnost	vrednost
LinearLayout1	Orientation	vertical
LinearLayout1	Layout height	fill_parent
LinearLayout1	Layout width	fill_parent
LinearLayout1	Background	*datoteka s sliko neba*
textView1	Layout height	wrap_content
textView1	Layout width	wrap_content
relativeLayout1	Layout height	wrap_content
relativeLayout1	Layout width	wrap_content
imageView1	Layout height	50dp
imageView1	Layout width	50dp
imageView1	Src	*datoteka s sliko balončka*

Programski del aplikacije

Aktivnost

- Aplikacije so sestavljene iz komponent, ki jih izvajalno okolje ustvari in požene.
- Pri izdelavi aplikacij z grafičnim vmesnikom je najpomembnejša komponenta ***Activity*** (*aktivnost*), ki je izdelana kot potomec razreda *Activity*
- Razvojno okolje navadno ustvari ogrodje različice razreda *Activity,* ko ustvarimo nov projekt.
- Z vidika razvijalca aplikacij je vstopna točka aplikacije povožena metoda *onCreate*.
  - Z drugimi besedami, v to metodo vključite programsko kodo, za katero želite, da se izvede najprej.

```
1  package AT.AT;
2
3  import android.app.Activity;
5
6  public class AT extends Activity {
7      /** Called when the activity is first created. */
8      @Override
9      public void onCreate(Bundle savedInstanceState) {
10         super.onCreate(savedInstanceState);
11         setContentView(R.layout.main);
12         //tu se začne vaša aplikacija
13     }
14 }
```

Naloge

- V nadaljevanju bomo sestavili aplikacijo na podlagi že pripravljenih delčkov kode.

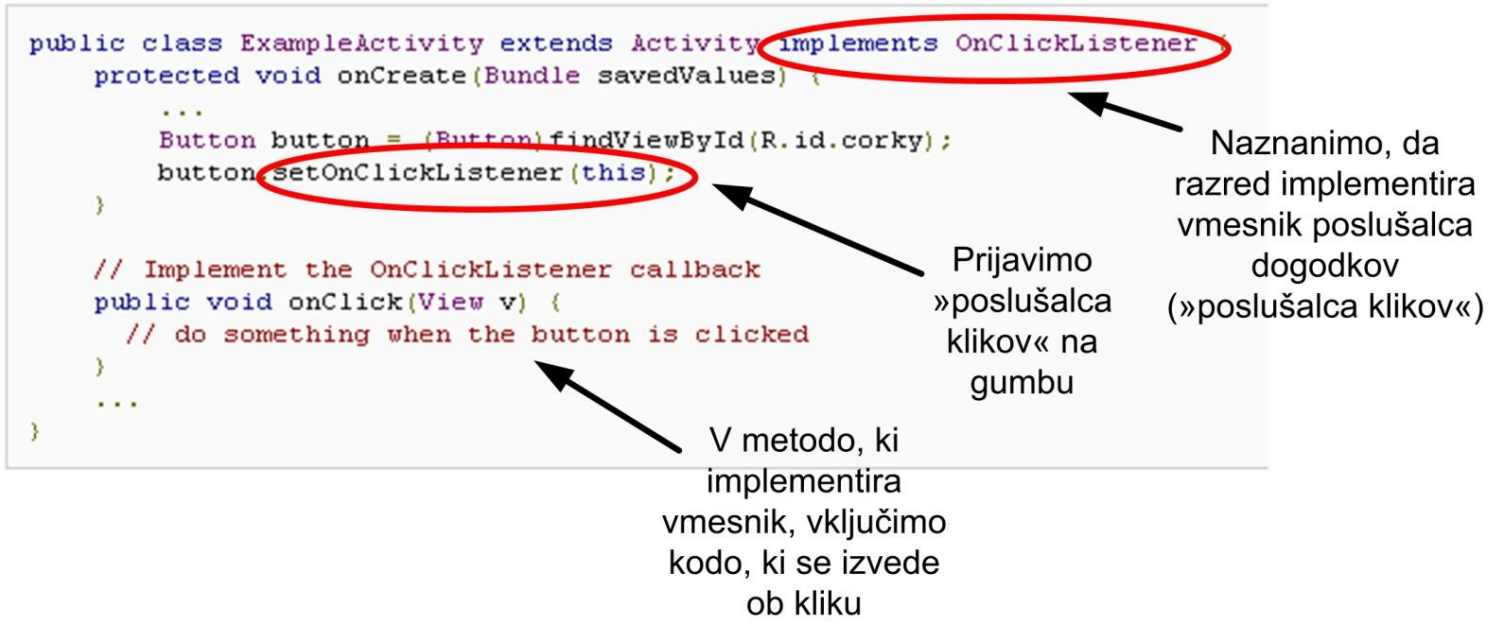
Vsak delček pripravljene kode vključite v ločeno metodo. Metode boste nato dopolnili in jih medsebojno povezali po spodnjih navodilih!

- Nadomestite vsebino razreda aktivnosti s kodo, ki je že pripravljena pod oznako [**aktivnost**]. Koda vsebuje ogrodje aplikacije.
- Aktivnost dopolnite tako, da se ob zagonu izvede inicializacija, nato pa se začne igra.
  - To storite tako, da kodo z inicializacijo [**inicializacija**] in začetkom igre [**zacniIgro**] najprej vključite v lastni metodi, ki ju nato pokličite iz metode onCreate.
- Kodo [**zacniIgro**] dopolnite tako, da sprožite premikanje balona [**premikanje**] in odštevanje časa [**odstevanje**].
  - V pripravljeni kodi imate s komentarji označena mesta, kjer morate kodo dopolniti.
- Preverite delovanje aplikacije v emulatorju.

Interakcija z uporabo dogodkov

- Aplikacija trenutno ne omogoča interakcije z uporabnikom, zato še ni uporabna
- Interakcija z uporabnikom je mogoča prek uporabe “dogodkov”
- »Dogodek« je pojem, ki je povezan z uporabnikovo interakcijo z določenim gradnikom uporabniškega vmesnika. Tipični dogodki so na primer:
  - dotik zaslona,
  - dvojni klik na ikono,
  - zasuk mobilnega terminala za 90 stopinj.
- Za izkoriščanje dogodkov v aplikaciji moramo **dogodke »prestreči«**
  - **Prijavimo »poslušalca« izbranega dogodka na izbranem gradniku.**
  - V argumentu metode, s katero prijavimo poslušalca, **navedemo, kje se nahaja upravljalnik dogodka**, ki ga prijavljamo;
  - **izdelamo »upravljalnik dogodka«.** Upravljalnik dogodka je del kode, v katerem je sprogramiran odziv na dogodek. Ta koda se izvede, ko se dogodek dejansko zgodi.
- Analogija z uporabnikovo interakcijo s spletnimi stranmi!

Primer uporabe dogodka



- Na podoben način lahko »poslušamo« dogodke tudi na drugih gradnikih grafičnega uporabniškega vmesnika.
- Imena metod, v katerih mora biti sprogramirana reakcija na posamezen dogodek, najdemo v dokumentaciji aplikacijskega programskega vmesnika.
  - http://developer.android.com/guide/topics/ui/ui-events.html
  - http://developer.android.com/reference/android/widget/package-summary.html

Naloge

- Prijavite poslušalca dotika na pogledu, ki vsebuje sliko balona.
  - Poslušalca prijavite na označenem mestu v [**inicializacija**].
- Izdelajte upravljalnik dogodka.
  - Upravljalnik naj bo izdelan kar v razredu aktivnosti.
- Sprogramirajte reakcijo na dogodek:
  - Povečajte števec zadetkov.
  - Sprožite zvočni efekt [**zvocniEfekt**].

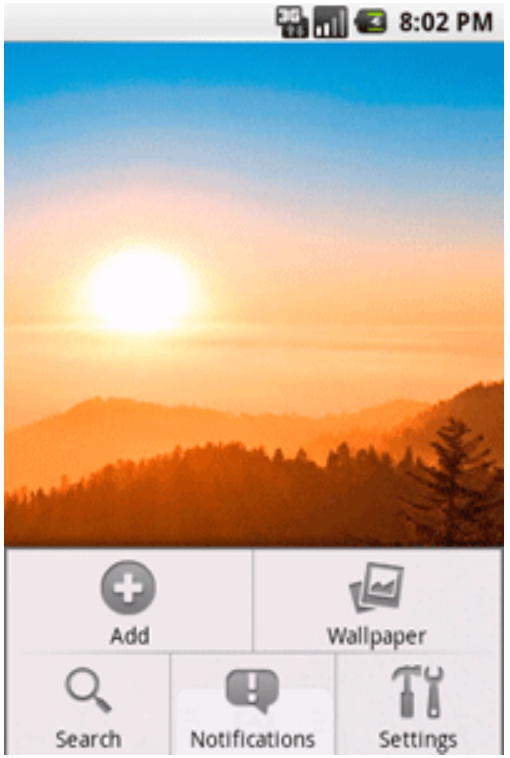
Meniji

- S pomočjo menijev lahko aplikaciji nastavljamo funkcije in lastnosti.
- Platforma Android ponuja že nekaj pripravljenih oblik menijev, ki jih lahko uporabimo.
- Najpogostejši je “options” meni, ki se prikaže ob pritisku na temu namenjeno tipko na tipkovnici mobilnega terminala.
- V ta meni lahko vključimo ikone in tekst za največ šest menijskih postavk. Kadar ima aplikacija veliko število nastavitev, ki so organizirane po tematskih področjih, lahko znotraj »options« menija uporabimo tudi »podmenije«
- Primer izdelave “options” menija najdete na

http://developer.android.com/guide/topics/ui/menus.html#options-menu

- Primer izdelave “podmenija” najdete na

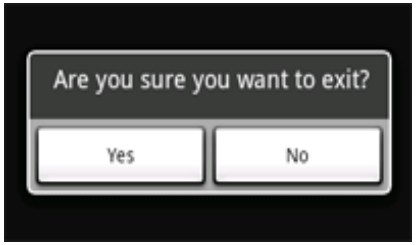
http://developer.android.com/guide/topics/ui/menus.html#submenu

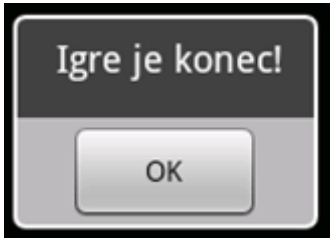


Dialogi

- Dialog je majhno okno, ki se pojavi v trenutni aktivnosti.
- Obstaja več vrst dialogov, ki so namenjeni za pozive, zbiranje podatkov ali obveščanju uporabnika.
- V naši aplikaciji bomo uporabili dialog z obvestilom (*alert*)
- Več o dialogih na

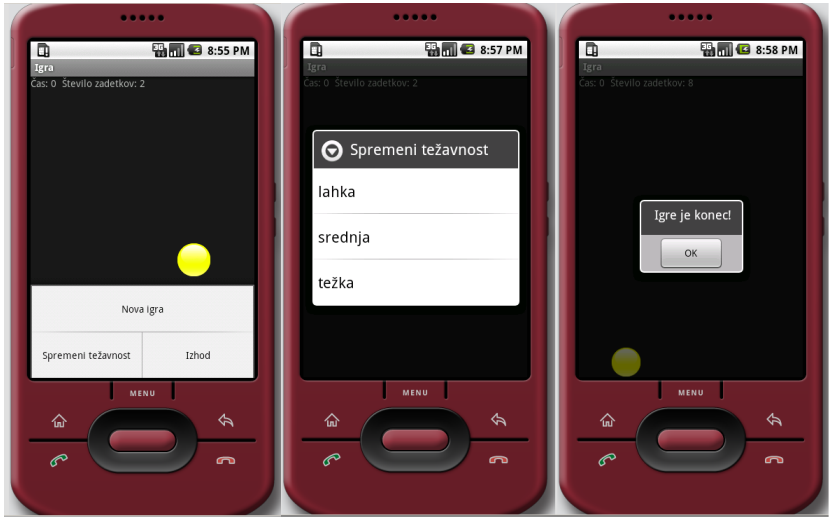
http://developer.android.com/guide/topics/ui/dialogs.html





Naloge

- Ustvarite menije. V dokumentaciji poiščite ime metode, v kateri ustvarimo “options” meni, in vanjo vključite kodo [**ustvariMeni**].
- V dokumentaciji poiščite ime metode, v kateri določimo, kaj se zgodi, ko je izbrana določena menijska postavka, in vanjo vključite kodo ([**izberiMeni**], [**koncajIgro**]).
- Dopolnite menije tako, da bodo podobni tistim na sliki.
- Ko se čas izteče [**odstevanje**], končajte igro [**koncajIgro**] in obvestite uporabnika [**dialog**].



REST Odjemalec

- RESTful storitve so zelo primerne za dostavo vsebine napravam, za katere je značilna omejenost razpoložljivih virov, kot so to pametni telefoni in tablični računalniki

- Primer: klic REST storitve za izpis točkovne lestvice v izdelani igri in vpisa zadnjega rezultata
  - HTTP zahtevo tvorimo v ločeni niti, ki teče "v ozadju" (background)
  - Ko dobimo rezultate, jih s pomočjo povratnega klica izpišemo
  - Vzorec uporabljamo pogosto, ko uporabljamo omrežje ali izvajamo računsko intenzivno opravilo

REST_Lestvica, REST_Rezultat

Naloga

- Povežite Igro s spletno storitvijo
  - Ko se igra konča, naj se rezultat pošlje spletni storitvi
  - V meni vgradite možnost prikaza lestvice najboljših igralcev na podlagi klica spletne storitve

- Potek
  - V manifestu projekta napovemo uporabo interneta (vrinite vrstico v rdečem)

```
<uses-sdk android:minSdkVersion="…" />
<uses-permission
   android:name="android.permission.INTERNET"/>
<application>
```

  - V paket z aktivnostjo prenesite razreda REST odjemalcev REST_Lestvica in REST_Rezultat
  - Na primernih mestih v razredu aktivnosti opravite klic REST odjemalcev z ustreznimi parametri

REST_Lestvica, REST_Rezultat

Naloga

- Preizkusite celotno infrastrukturo izdelano na vajah
  - Preizkusite Igro v emulatorju
  - Preizkusite spletno aplikacijo, ki je dostopna na http://212.235.190.222:8080/SeminarWebREST/index.jsp

Še nekaj opozoril pri razvoju

- Če razvijate aplikacijo za Android, ki uporablja klice REST storitve, dostopne na lokalnem računalniku, upoštevajte sledeče
  - Če boste REST storitev poganjali iz Eclipsa (kot na vajah), potrebujete dva ločenega delovnega prostora: enega za storitev (poženete jo iz Eclipse EE), drugega za mobilno aplikacijo (poženete iz ADT)
  - Domensko ime "localhost" v androidni aplikaciji se ne sklicuje na vaš računalnik ampak na androidno napravo (ali pa njen emulator)
    - Če želite iz emulatorja dostopati do lokalnega računalnika, v URL namesto "localhost" uporabite IP naslov "10.0.2.2"

Literatura

- Nadaljnje informacije za delo s platformo Android dobite v sledeči literaturi
  - Uradna spletna stran za razvijalce aplikacij za Android http://developer.android.com/guide/index.html
  - Knjiga za začetnike in za naprednejše razvijalce: R. Meier: *Professional Android Application Development*, Wiley Publishing, 2009
  - "Calling RESTful services from your Android app" **http://www.techrepublic.com/blog/app-builder/calling-restful-services-from-your-android-app/1076**