Riadenie vykonávania

- if-else (mali sme už minule)
- return (mali sme už minule)
- while
- do-while
- for
- break/continue
- switch/case

If-else (C, Java):

```
if (i < 10) {
//rob niečo

} else {
//rob niečo iné

}
```

Return (pre funkciu v C / metódu v Jave):

```
int getArea() {
   return (hodnota);
}
```

While / do-while (Java, C) cyklus

- rozdiel je v tom, kedy sa vyhodnotí výraz, ktorý rozhoduje o opakovaní

```
while (vyraz) {
     //príkzy sa vykonávajú pokým ma výraz hodnotu true
}
```

```
do {
     //príkzy sa vykonávajú pokým ma výraz hodnotu true
} while (vyraz)
```

For (Java, C) cyklus

```
for (inicializácia; podmienka na opakovanie; inkrementácia) {
     //príkzy sa vykonávajú pokial podmienka opakovania = true
}
```

```
for(int i=1; i<11; i++){
    System.out.println("Count is: " + i);
}
```

Break/continue (Java, C)

- cyklus (while/do-while, for) pri vykonaní príkazu break sa okamžite končí
- cyklus (while/do-while, for) pri vykonaní príkazu break sa okamžite opakuje od začiatku

```
for(int i=1; i<11; i++){
    if(podmienka1) {
       // ak podmienka1 je splnená potom sa okamžité končí celý cyklus
       break;
    }
    if(podmienka2) {
       // ak podmienka2 je splnená potom sa okamžité opakuje celý cyklus od začiatku
       continue;
    }
    System.out.println("Count is: " + i);
}
```

Switch-case (Java, C)

- opakované if-else if-else pri menšom počte podmienok je kompaktné a ešte v poriadku

```
if(premenna==hodnota1) {
   //rob nieco
} else if (premenna==hodnota2 || premenna==hodnota3) {
   //rob to isté pre hodnotu2 aj hodnotu3
} else {
//pre všetky ostatné hodnoty rob toto
      break;
}
```

- pri vačsom počte je však neprehľadné a pomalé
- náhrada opakovaného if-else if-else ... je switch-case-break (pre niektoré typy hodnôt)
    - primitívne typy: byte, short, char, and int
    - enumeračné typy, trieda String, Byte, ...

```
switch (hodnota premennej) {
   case hodnota1:
      //rob nieco
      break;
   case hodnota2:
      //rob to isté čo pre hodnotu3
   case hodnota3:
      //rob nieco
      break;
   default:
      //pre všetky ostatné hodnoty rob toto
      break;
}
```

Jazyk Java – modulárne programovanie (opakovací slide)

- vytvoríme si viac modulov **- každý bude predstavovat jednu triedu (tj. teraz modul=trieda)**
  - budeme používať modifikátory: static, public/private, final
  - moduly (triedy) môžu mať svoje konštanty, premenné (atribúty)
  - moduly (triedy) môžu mať svoje funkcie (metódy)
- jednoduchý program sa typicky skladá zo súborov <trieda_A>.java, <trieda_B.java>
  - jedna trieda obsahuje funkciu „main“ (toto sme už mali minule):

```
public static void main(String[] args) {
...
}
```

- Príklad - hello world [pokus2_moduly]

Rozšírené modulárne programovanie – balíky

- Balíky (packages) umožňujú lepšie štrukturovať väčšie programy, dôvody
  - opakovaná použiteľnosť kódu
  - lepšia organizácia kódu
  - riešenie konfliktov pomenovaní (napr. System.out.println, java.sql.DriverManager.println, ...)
- Kľúčové slovo „package“ deklaruje príslušnosť modulu (triedy) k danému balíku
- Použitie iných balíkov sa deklaruje kľúčovým slovom import
  - Napr. deklarácia použitia len jednej triedy: “import java.util.ArrayList;”
  - Napr. deklarácia použitia celého balíka: “import java.util.*;”
  - Okrem java.lang ostatné štandardné balíky Javy treba explicitne importovať
    - My sme už takto napr. použili java.lang.System.out.println, java.lang.Math.pow
- Štruktúra a dokumentácia všetkých štandardných balíkov sa dá prehliadať napr. na http://docs.oracle.com/javase/7/docs/api/

Java OOP – úplný základ

- Čo je OOP?
  - Programovanie pomocou objektov
  - Program sa uskutočňuje ako interakcia objektov
- Čo je objekt?
  - objekt je entita, ktorá má:
    - stav – zahŕňa všetky vlastnosti objektu (atribúty) a ich hodnoty
    - správanie – ako objekt koná a reaguje v zmysle zmeny stavu a operácií, ktoré poskytuje
    - identitu – jednoznačná identifikácia objektu
    - typ – typ objektu (trieda)
- Trieda
  - Určuje typ objektu a tým aj jeho správanie
  - Trieda má názov a telo
    - Class nazovTriedy {/* tu je telo */}
  - Trieda môže obsahovať
    - Atribúty (údaje)
      - Primitívneho typu (char, byte, short, int, long, float, double, boolean, void)
      - Referencie (odkaz) na objekty
    - Metódy (operácie)
      - Podobné funkciám v C, ak nevracajú žiadnu hodnotu typ je `void`

```
typ nazovMetody(/* parametre */) {
   /*TELO METODY*/
   return nieco; /*alebo aj cisto „return“ ak typ je void*/
}
```

    - Triedy (vnútorné triedy)

Java OOP – úplný základ (2)

- jednoduchý program sa typicky skladá zo súborov <trieda_A>.java, <trieda_B.java>
- jedna trieda obsahuje funkciu „main“ (toto sme už mali minule):

```
public static void main(String[] args) {
...
}
```

- čo znamená „public“? (je to modifikátor prístupu)
  - tu sa dostávame k dôležitému pojmu „zapuzdrenie“ (enkapsulácia)
    - to čo má zostať v triede skryté, treba odpovedajúco označiť (modifikátor „private“)
    - prístup / rozhranie tvoria vybrané metódy (v tomto prípade „main“)
- čo znamená static? (tiež je to modifikátor)
  - metóda patrí triede, môžeme ju volať skôr, než vôbec vznikne nejaký objekt z danej triedy
  - dá sa aplikovať na atribúty, metódy, triedy ...
- môžeme zapuzdrenie vnárať ?
  - potrebujeme ďalší pojem „kvalifikácia“, spôsob ako sa dostávame k triede/objektu a jeho metódam/atribútom
  - potrebujeme pojem „vnútorná trieda“
    - príklad: „a=triedaA.triedaB.getNieco()“ ...
- môže byť v jednom *.java súbore viac tried?
  - Áno ... ukážeme si to ...
- čo je konštanta (napr „final int a =5;“) a čo literál (napr. int a = 123);

Stručný životný cyklus objektu

- Vytvorenie objektu
  - objekt vytvoríme pomocou operátora new
  - napr: Student Jano = new Student();

```
String s = new String („text“);
```

- Interakcia Objektov
  - objekty posielajú správy – t.j. volajú metódy tých objektov, ktorým chcú poslať správu
    - xx=objA.getAge()
- priradenie objektu
  - priraďujú sa referencie nie hodnoty
    - napr: Student mudryStudent=Jano
    - mudryStudent aj Jano ukazujú na ten istý objekt
    - zmena cez hociktorú z týchto referencií vplýva ne ten istý objekt
- objekt ako parameter metody
  - predáva sa odkaz, nie samotný objekt
- Zánik objektu
  - V jave sa objekty nerušia explicitne
  - objekty, na ktoré sa nikto neodvoláva, ruší zberač smetí (garbage collector)

```
{
   String s = new String („text“);
} // tu už objekt nie je dostupný, pravdepodobne už zanikol
```

Metódy a atribúty triedy a objektu

- čo patrí triede a čo objektu?
  - Statické metódy a statické atribúty patria triede (existujú ako súčasť triedy)
  - „private static final int a=5;“ // konštnta
  - „private static int b=5;“ // inicializovaná premená
- Objekt = „inštancia triedy“
  - Líši sa od triedy hlavne tým, že obsahuje svoju vlastnú kópiu nestatického obsahu
  - zdieľa s triedou statický obsah (vie triedu zastúpiť)
    - volanie statických metód „cez triedu“ a „cez objekt“ má ten istý efekt
  - dynamický obsah sa konštruuje → „konštruktor“
  - „private int mVlastnostXY; //zvyčajne napĺňa konštruktor“
- Špeciálna metóda „konštruktor“
  - má taký istý názov ako trieda
  - vyvolá sa pri vytváraní objektu, má za cieľ zinicializovať objekt

Príklad:

- Netbeans prostredie
  - [pokus3_objekty]

Typy premenných – sumarizácia (opakovanie)

- V Jave sú 3 typy premenných
  - Lokálne premenné
    - Sú deklarované/definované v metódach a blokoch { }, len tam sú viditeľné
    - Vznikajú na zásobníku pri vstupe do metódy/bloku zanikajú pri vystúpení z metódy/bloku
    - Lokálne premenné môžu mať iba modifikátor “final”
      - Na rozdiel od C nemôžu byť static (namiesto toho sa majú používať premenné objekty/triedy)
  - Premenné objektu (inštancie triedy)
    - Deklarované v triede mimo metód
    - Vytvárajú sa a zanikajú spolu s objektom
    - Viditeľné v rámci objektu
    - Môžu používať modifikátory
    - Majú implicitne definovanú hodnotu (0, false, null)
  - Premenné triedy
    - Ako premenné objektu ale s modifikátorom „static“
    - Existuje v pamäti iba raz
    - Vytvorené keď štartuje program a zanikajú keď program
    - Môžu používať modifikátory
- Premenné mimo vyššieuvedených java nepovoľuje (napr. na začiatku modulu)

Typy údajov v Jave (opakovanie)

- Primitívne typy (byte, short, int, long, float, double, char, boolean)
- Objekty

Obaľovacia trieda pre primitívne typy

- Pre každý primitívny typ je definovaná obaľovacia trieda (názov začína veľkým písmenom)
  - char – Character
  - int – Integer
  - …

Java zásobník a hromada [x]

- Zásobník (stack, LIFO) – údaje s krátkou dobou životnosti
  - lokálne premenné s primitívnymi typmi
  - lokálne referencie na objekty
  - každé vlákno má vlastný zásobník, t.j. zásobník je „thread safe“
- Hromada (heap)
  - globálne adresovateľné entity
    - triedy
    - objekty
      - „garbage collector“ pravidelne zbieha a maže objekty na ktoré už neexistujú referencie
      - nie sú „thread safe“

Triedy - Modifikátory

Modifikátory prístupu (zatiaľ sme v tom istom balíku)

- **Bez modifikátora/public** – k atribútu/metóde vedia všetci R/W pristupovať
- **Private** – k atribútu/metóde môže pristupovať len metóda aktuálneho objektu

Ostatné modifikátory

- **Final** – už sa nemôže modifikovať (vysvetlené minule)
- **Static** – statická metóda/atribút (vysvetlené minule)

Triedy a metódy – „this“

- this - vráti referenciu na aktuálny objekt
  - napr. na rozlíšenie medzi parametrom metódy (napr. id) a atribútom metódy (this.id)
  - v statickej metóde sa nedá použiť

Triedy a metódy – preťaženie metód

- viaceré metódy v tej istej triede môžu mať rovnaký názov ak sa líšia v signatúre
  - signatúra = názov operácie a typy všetkých argumentov
  - návratová hodnota sa nedá použiť na rozlíšenie
- toto platí aj pre konštruktory

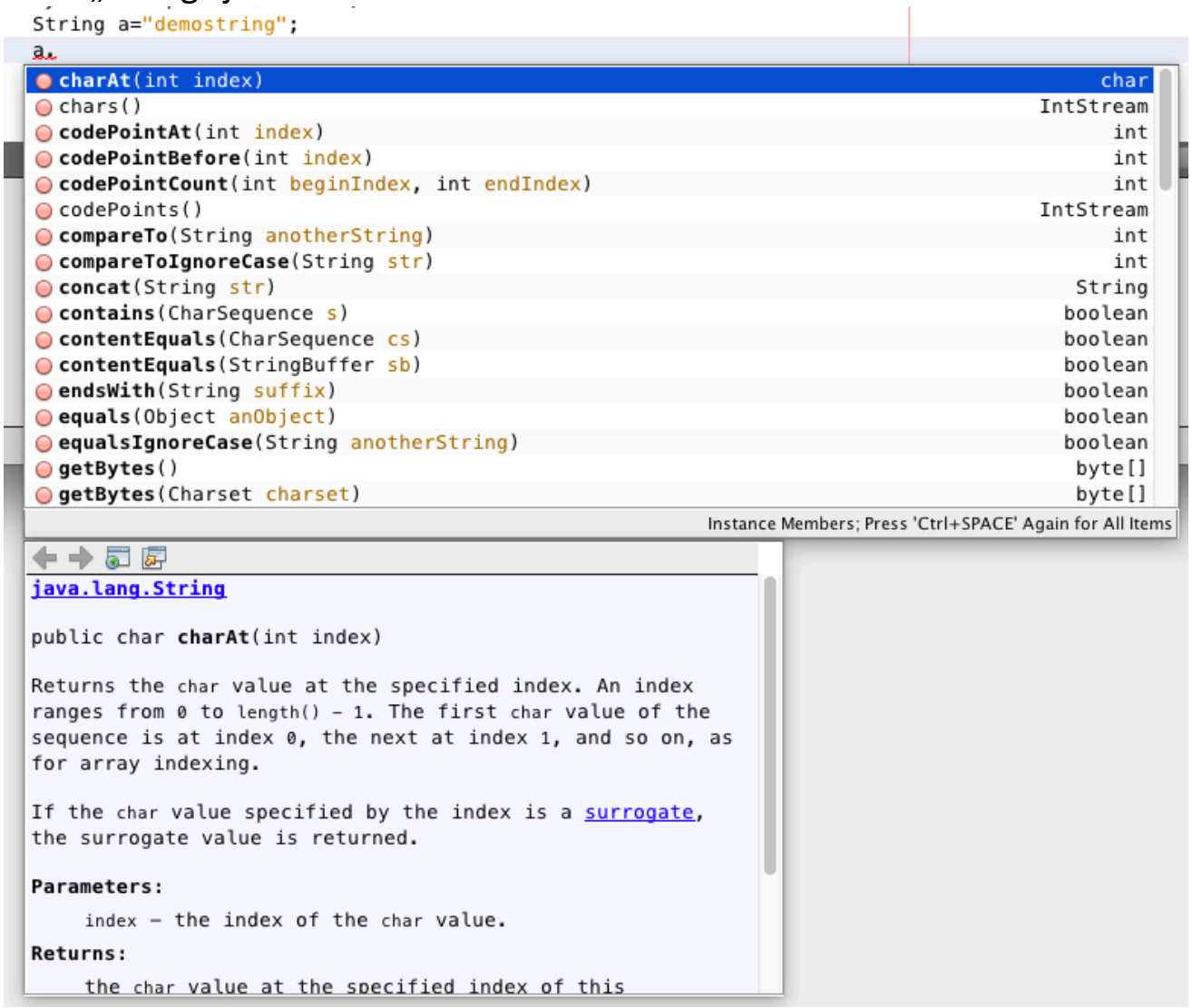
```
class Kruh {
   private double R;
   public Kruh (double myR) {R=myR;};
   Kruh () {this(5.0);};
}
```

Inicializácia

- pri objektoch musia byť prítomné konštruktory
  - ak programátor explicitne nenapíše konštruktor, prekladač poskytne implicitný (prázdny) konštruktor
- pri atribútoch – ak nie je explicitná inicializácia, nastavia sa implicitné hodnoty
- pri premenných v metódach – musí byť explicitná inicializácia
- blok príkazov v triede ohraničený
  - { } – vykoná sa pri inicializácii každého nového objektu
  - static { } – vykoná sa pri inicializácii triedy
- Príklad inicializácií [Prednaska03.java]

Reťazce a práca s reťazcami

- „String“ je trieda s veľa metódami ....



Reťazce 2

- Poznámka k reťazcom a ich inicializácii
    - Z hľadiska úspory pamäte je lepšie vytvárať reťazce bez „new“ a dôverovať kompilátoru ...

```
String a = new String("text");
String b = new String("text");
String c = "text";
String d = "text";
System.out.println(a == b); // false, referencie na objekty sú rôzne
System.out.println(c == d); // true, odkazujú na ten istý object(optimalizácia javy)
```

Porovnanie dvoch retazcov, či sú identické

- Metóda „equals“ triedy String
- if(a.equals(b)) {...}
- ak sú dva stringy null, potom sú rovnaké (c==d) je true,
    - c.equals(d) nemôžeme použiť, lebo objekt je null

Operátor +

- stringy môžeme spájať pomocou +

```
String c = "text1";
String d = "text2";
c+=d;
System.out.println(c);
```

- samozrejme môžeme reťazce spájať aj pomocou metódy „concat“

Formátovanie

```
String fs = String.format("Hodnota floatu je %f, hodnota integeru”+
                          “je %d, and retazca je %s", floatVar, intVar, stringVar);
System.out.println(fs);
```