Mer klasser och objekt

Vi kikar in i klasserna, och tar en titt på några av javas inbyggda

Kodkonvention

- Klasser påbörjas med stor bokstav (i övrigt små utom om flera ord då Inledande bokstav i övriga ord också görs stor)
  - `AClass`
- Metoder/attribut inleds med liten bokstav (i övrigt små utom om flera ord då inledande bokstav i övriga ord också görs stor)
  - `aMethod(), aVariable`
- Konstanter Bara stora bokstäver. Ord åtskiljs med _
  - `PI`
- Följer man dessa konventioner så kommer ens egna klasser att ha samma namngivning som javas inbyggda.
  - Vilket underlättar läsbarhet och samarbete.

Synlighet för attribut, metoder och klasser

Attribut eller metod	Nås i samma klass	Nås i samma paket	Nås i alla subklasser	Nås överallt
`private`	ja	nej	nej	nej
default	ja	ja	nej	nej
`protected`	ja	ja	ja	nej
`public`	ja	ja	ja	ja

Överlagring

- Vi kan ha flera metoder/konstruktorer med samma namn bara dessa har olika signatur
- ***signatur*** : metodens namn och parametrar (antal,typ och ordning)
  - Observera att vi inte kan ha metoder som bara skiljer sig åt beroende på returtyp
- Vilken version av metoden som används vid anrop beror på de aktuella parametrarna
- Ex:
  - `public int aMethod(int x) {...}`
  - `public int aMethod(int x, int y){...}`
  - `aMethod(1)`
  - `aMethod(1,2)`

Överlagrade konstruktorer

- Vill man att en av konstruktorerna i en klass ska anropa en annan (för att få en och samma kod på bara ett ställe) så gör man detta mha `this()`
- `this()` måste stå först i konstruktorn
- Ex:

```
public Square()
    this(60,50);
}

public Square(int xPositionP,int yPositionP){
    size = 30;
    xPosition = xPositionP;
    yPosition = yPositionP;
    color = "red";
    isVisible = false;
}
```

Mer this

- I en metod refererar referensen `this` till det objekt vars metod för tillfället körs
- `this` kan användas till att referera till godtyckligt attribut/metod för objektet
- Används tex då parametrar har samma namn som attribut för att särskilja dem.
- Ex

```
public Square(int xPosition, int yPosition){
    size = 30;
    this.xPosition = xPosition;
    this.yPosition = yPosition;
    color = "red";
    isVisible = false;
}
```

Strängar

- Representeras i Java av en klass.
  - `String`
- Svarar mot en sekvens av tecken.

```
String str=”abc”;  //finns även andra sätt skapa strängar
```

- Operationer på strängar:
  - + Strängkonkatenering
  - Ex:
    ```
    "hej" + "då" => "hejdå"
    "Antal: "+1 => "Antal: 1"
    ```
  - `charAt(int)` tar fram bokstaven på en given position (numrerat från 0)
  - `length()` tar fram antal bokstäver i strängen
  - `equals(String)` kollar om två strängar innehåller samma tecken
  - mm. (se API beskrivningen: https://docs.oracle.com/en/java/javase/11/docs/api/java.base/java/lang/String.html)

Enkel output

- Kan tex göras mha klassen `PrintStream`
- Fördefinierade ***standardströmmar***:

ström	**syfte**
`System.out`	skriva vanlig text
`System.err`	skriva felmeddelanden

`PrintStream`

- Huvudsakligen `print` och `println` metoder som skriver ut värden.
- `println` gör radbrytning efter utskrift
- `print` och `println` är överlagrade för alla primitiva datatyper, `String` och `Object`

➡`print` och `println` finns för alla tänkbara datatyper

`PrintStream`

- Huvudsakligen `print` och `println` metoder som skriver ut värden.
  - `println` gör radbrytning efter utskrift
  - `print` och `println` är överlagrade för alla primitiva datatyper, `String` och `Object`

Kontrollstrukturer i Java

- Rätt lika de som de som finns i C med vissa småskillnader
  - `if(villkor)`
  - `if(villkor) ... else`
  - `while(villkor)`
  - `do ... while(villkor)`
  - `for(init;villkor;förändring)`
- Villkoren ovan ska alltid vara av typen `boolean`

`switch`-satsen

```
switch (expression) {
    case value1:
        sats1;
        break;
    case value2:
        sats2;
        break;
    …
    default:
        satsN;
}
```

OPTIONAL

Uttrycket måste vara heltal, tecken, String eller enum
Motsvarar nästade if-satser

`static`

- Attribut och metoder kan deklareras `static`
- Associerar attribut och metoder med klassen snarare än objektet
- Avsteg ifrån objekts-tanken
  - Bör därför användas bara där det verkligen behövs
- Undvik under denna kurs (utom för `main`) då ni ska lära er objektorientering.

Statiska attribut

- Normalt har varje objekt eget minnesutrymme
- `static` gör variabeln gemensam för alla objekt i klassen (dvs den delas av alla)

```
private static int count;
```

- Kallas ibland ***klass variabler***

Statiska metoder

- Statiska metoder anropas via klassnamnet
- Man behöver inte skapa ett objekt först
- Därför kallas de också ***klass metoder***

- Exempel: Klassen `Math` i paketet `java.lang` innehåller statiska matematiska operationer

```
Math.abs(num)       -- absolutbeloppet
Math.sqrt(num)      -- kvadratrot
Math.random()       -- slumptal i intervallet [0.0...1.0)
```

Statiska metoder

- Statiska metoder får inte referera till instansvariabler (eftersom de instansieras först när ett objekt skapas)
  - Får bara referera statiska variabler eller lokala variabler (som existerar oberoende av objekt)
- Även metoderna är specifika för objekt
  - Får bara anropa statiska metoder (om den inte gör det via en instans)

Metoden main

- Metoden

```
public static void main(String[] args)
```

  Anropas automatiskt då man vill starta ett javaprogram

- Metoden måste se ut just som ovan (utom args som är en godtycklig identifierare)
- Denna metod får sedan se till så att de objekt som behövs av programmet skapas och exekverar metoderna som kör igång programmet

Konstanter

- Som variabler fast deras värde får inte ändras
- Markeras med reserverade ordet `final`
- Exempel:

```
final double PI = 3.14159;
final char NEWLINE = '\n';
```

- Fördelar
  - Vettiga namn istället för konstiga siffror eller tecken.
  - Lättare att förstå koden.
  - Enklare att uppdatera.
- Rätt ofta även deklarerade `static` så att bara en konstant finns för hela klassen.