OOP – Teoretiske emner

Dato

Teoretiske emner: en klasses opbygning og constructors

Klasse

En klasse er en definition på, hvordan et instans af den klasse, altså et objekt, skal opføre sig, og hvad det kan indeholde af værdier og referencer.

Som det første i klassen defineres klassen attributter, der er dataelementer, som programmet kan arbejde på. Hver attribut har en acces modifier, en type og et navn.

Constructor

En klasses “Constructor” bliver kaldt, når der oprettes et nyt instans af en klasse. Constructoren bestemmer, hvordan objektet skal opbygges til at starte med. F.eks. bestemmer den, hvilke attributter objektet starter med at have og deres værdi.

En constructor kan også have parametre, så en programmør kan indsætte nogle begyndelsesbetingelser, når et objekt bliver lavet. Har man f.eks. en person klasse, så kan nogle indledende parametre være, personens CPR-nummer, hvor personen bor osv.

Der kan også være flere forskellige constructors med forskelligt antal parametre, som så med forskellig varians kan bestemme, hvordan objektet defineres som start.

Definerer man en variabel (attribut) med værdi inden en constructor, altså direkte i klassen, er denne variabel tilgængelig for alle instanser af den klasse, uanset hvilken constructor, der er blevet brugt til at lave det objekt. Det kalder man i Python klassevariabler.

CprNR

Teoretiske emner: simple datatyper og Strings i Java

Simple datatyper

Data Type	Size	Description
byte	1 byte	Stores whole numbers from -128 to 127
short	2 bytes	Stores whole numbers from -32,768 to 32,767
int	4 bytes	Stores whole numbers from -2,147,483,648 to 2,147,483,647
long	8 bytes	Stores whole numbers from -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Stores fractional numbers. Sufficient for storing 6 to 7 decimal digits
double	8 bytes	Stores fractional numbers. Sufficient for storing 15 decimal digits
boolean	1 bit	Stores true or false values
char	2 bytes	Stores a single character/letter or ASCII values

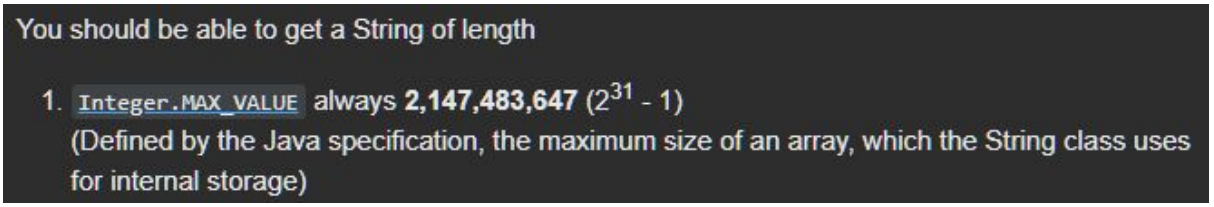
Simple datatyper, er typer, som ikke er en klasse (objekt af klasse), og som har en fast størrelse. Disse datatyper kan man ikke kalde metoder direkte på, da de ikke har nogen referencer til nogen klasse. De er nemlig call by value.

For at kunne bruge metoder, som er designet til de simple datatyper, skal man bruge nogle klasser, som er lavet til dem. Heldigvis kommer de indbygget i Java. Når man arbejder med ints, så er der en klasse Integer, som kan hjælpe med at bearbejde integers. Med denne kan man f.eks. konvertere til en string ved Integer.parseInt(int integer).

Strings

Strings er næsten en simpel datatype, men så alligevel ikke. String er en klasse, og en string kan også variere meget i størrelse. En string angives ved citationstegn (“ ”).

Strings har en maks størrelse, men den er meget større end de simple datatyper


You should be able to get a String of length

1. `Integer.MAX_VALUE` always **2,147,483,647** (2^{31} - 1)
   (Defined by the Java specification, the maximum size of an array, which the String class uses for internal storage)

I og med at String er en klasse, så kan man også direkte kalde metoder på strings, hvilket man ikke kan for de simple datatyper. Metoder som vi har brugt i undervisningen er f.eks. charAt(), equals(), indexOf().

StringTokenizer

Teoretiske emner: metoder i Java, call by value, call by reference

Metoder

Metoder er kode som kun køre når de kaldes. En metode består af en access modifier, en returnerings værditype og en identifier (navn). Metoder kan tage data i form af parametre, men behøver ikke have nogen parametre. Metoder bruges til at tage et stykke kode og afgrænse det. Det kunne være noget kode, som blev brugt flere steder i programmet, som man så tager ind i en metode, så man ikke hver gang skal skrive det samme kode men blot lave et metodekald.

Access-metoder er set- og get-metoder, der giver adgang til private eller protected attributter.

Call by value

Call by value er for de simple datatyper. Hver gang man har en variabel med en simpel datatype, så bliver den gemt. Hvis den værdi så ændres på, så er det selve den værdi, som ændres.

F.eks. har man en variabel A=2 og man laver en kopi af den B=A, så vil begge variable være gemt med værdien 2. Hvis jeg så ændrer A til at have værdien 4, så vil B stadig have værdien 2, da det er den **VÆRDI** som er gemt til variablen B.

Call by reference

Call by reference er for objekter af klasser. Når man har et instans af en klasse, så er den variabel en reference til **ET** sted i hukommelsen, hvor selve objektet er gemt. Når så man tager det objekt ind i en metode som et argument, så er det ikke selve objektet, som metoden får med men en reference til det sted, hvor selve objektet er gemt i hukommelsen. Hver gang der så på et objekt bliver kaldt en metode, bruger Java referencen til objektet for at gå ind og hente metoden fra objektet i hukommelsen.

Fjernvarme

Teoretiske emner: arrays – anvendelse, fordele og ulemper

Arrays

Arrays bruges til at gemmen flere værdier i en enkelt variabel i stedet for at oprette separate variable for hver værdi. Et array erklæres med en type og firkant parenteser “[ ]”. I firkant parentesen skrives en int, der fortæller hvor mange pladser der er i arrayet. Den mest simple måde at tilføje værdier til arrayet er ved at sætte arrayet lig med en liste af kommaseparerede værdier. Hvis det ønskes at indsætte en værdi på en bestemt plads i arrayet, så sættes arrayets navn med firkant parentes, der indeholder pladsen, lig med værdien der skal indsættes.

ArrayList

Hvis man gerne vil gemme en liste af noget, men men ikke ved, hvor mange elementer, der kommer til at være brug for, kan man bruge ArrayList. Med ArrayList skal man ikke starte med at definere, hvor mange elementer, der må være i, man kan blot, når der skal et nyt element i Arrayet bruge metoden “add()” for at tilføje et nyt element til listen. For ArrayList skal man ikke bruge “[]” for at hente værdier heller. For at hente en værdi af en ArrayList, bruges metoden “get(int index)”, som returnerer værdien af et element i listen med det index, man specificerer.

Kunstværker

Teoretiske emner: access modifiers, arv og andre relationer mellem klasser og objekter

Access modifiers

Der findes i Java fire forskellige access modifiers, default (ingen ting), public, private og protected.

	default	private	protected	public
Same Class	Yes	Yes	Yes	Yes
Same package subclass	Yes	No	Yes	Yes
Same package non-subclass	Yes	No	Yes	Yes
Different package subclass	No	No	Yes	Yes
Different package non-subclass	No	No	No	Yes

Default modifieren, altså hvor man ikke erklære nogle modifier, betyder at attributter, metoder og klasser er tilgængelig for alle andre klasser i den samme mappe (package).

Med **public** modifieren er attributter, metoder og klasser tilgængelige fra alle andre klasser.

Med **private** modifieren er attributter, metoder og klasser kun tilgængelige for klassen egne metoder.

Med **protected** modifieren er attributter, metoder og klasser kun tilgængelige for klassen egne metoder, som private, plus klasser som arver fra aktuel klasse.

Arv og andre relationer (interfaces eller importering/samme package) mellem klasser og objekter

Arv

Arv er en klassestruktur. En sub-klasse arver attributter og metoder fra super-klassen. For at en sub-klasse kan arve fra en super-klasse bruges “extends”. En sub-klasse kan have metoder med samme signatur, men forskellig implementering end super-klassen, dette kaldes polymorfi.

Super-klassen “Husdyr”.

```java
public abstract class Husdyr {
    protected String navn;
    protected String livret;

    public Husdyr(String etN, String enL) {
        navn = etN;
        livret = enL;
    }

    public String getNavn() {
        return navn;
    }

    public String getLivret() {
        return livret;
    }

    public String givLyd() {
        return "Ubestemt dyrelyd";
    }
}
```

Sub-klassen “Kat”.

Der er polymorfi i forhold til metoden “givLyd()”. Denne overskrives af sub-klassen egen metode.

```java
public class Kat extends Husdyr {
    private int antalMysFanget;

    public Kat(String etN, String enL) {
        super(etN, enL);
        antalMysFanget = 0;
    }

    public int getAntalMysFanget() {
        return antalMysFanget;
    }

    public String givLyd() {
        return "Miau";
    }
}
```

Interfaces

Interfaces er en outline for, hvilke ting, som skal med i en klasse. Hvis man f.eks. gerne vil lave en vanddunk klasse, men men ikke ved, hvad for nogle ting en vanddunk kan eller skal indeholde, så kan man implementere et vanddunk interface (hvis man går ud fra, at der er lavet et sådant interface), som fortæller en, hvilke metoder, man skal implementerer. I et interface kan man have defineret nogle attributter. Atributternes værdi kommer med til klassen som implementerer interfacet, men klasserne i interfacet er kun deklareret, man skal selv implementere den kode som skal køres. Det kan tydeligt ses i en klasse med et interface implementeret, hvilke metoder er fra interfacet og hvilke er for klassen selv, da dem fra interfacet skal have et @override.

```
1  interface WaterBottleInterface {
2      String color = "Blue";
3
4
5      void fillUp();
6      void pourOut();
7  }
8
9  public class InterfaceExample implements WaterBottleInterface {
10
11     public static void main(String[] args) {
12         System.out.println(color);
13
14         InterfaceExample ex = new InterfaceExample();
15         ex.fillUp();
16     }
17
18     @Override
19     public void fillUp() {
20         System.out.println("It is filled");
21     }
```

En ulempe ved et interface kan være, at man SKAL implementerer alle de metoder, som er deklareret i interfacet, man kan ikke bare nøjes med at implementerer f.eks. 2 af de 10 metoder, som var i det interface.

Importering/samme package

En anden relation, som ikke direkte er en relation men mere en brug af én klasse i en anden klasse er importering / brug i samme mappe (package). Hvis man har en klasse, der indeholder nogle generelle funktioner, som man gerne vil bruge i en anden klasse kan man importere den, hvis den er i en anden package, eller bare bruge den, hvis den ligge i samme package. På denne måde kan man lave instanser af forskellige klasser i en anden klasse.