TDDC77 Objektorienterad Programmering

Föreläsning 6

Sahand Sadjadee
IDA, Linköpings Universitet

Outline

- Överlagring (Overloading)
- Åtkomt (Access)
- Inkapsling (Encapsulation)
- 2D-arrays
- Standard-bibliotek
- Wrapper klasser

Överlagring(Overloading)

Klassen calculator

```
class Calculator{
  private double result;

  public Calculator(){
  result = 0;
  }

  public Calculator(double mem){
  result = mem;
  }

  public double add(double a){
    result += a;
    return result;
  }

  public double add(double a, double b){
    result = a + b;
    return result;
  }

  public void reset(){
    result = 0;
  }
}
```

Signatur

```
public static int parseInt(String s){}
```

- En <u>konkret</u> metod deklareras med hjälp av fyra komponenter
  - Returtyp eller void
  - Metodens namn (identifierare) som ska börja med en liten bokstav
  - Parameterlistan med noll eller fler parametrar.
  - Metodens kropp inom måsvingar
- Metodens signatur består av metodens namn och parameterlistan.

```
public double getMyFundsFromBank(String bankName)
```

signature is method name + parameters only

Överlagring

- En metod överlagrar en annan metod, i samma klass, om de har samma namn (identifierare) men olika signaturer.
- Returtypen spelar ingen roll för överlagring vilket innebär att två metoder med olika returtyper men samma signatur inte är tillåtna (kompilatorn kan inte skilja på dem).

Åtkomst(Access)

Åtkomst

- Man kan sätta åtkomst till klassmedlemmar, fält och metoder.
- En klassmedlem kan vara:
  - `private`: bara metoder i klassen får använda den.
  - `public`: Alla metoder i alla klasser får använda den.
  - `default`: Alla metoder som tillhör till klasser som ligger i samma paket, mapp, får använda den. Default är inget nyckelord och gäller när ingen access-modifierare används explicit i deklarationen.
  - `protected`: Vi pratar om det senare i kursen.
- Åtkomst kommer alltid först i deklarationen.

Åtkomst

Syntax:

```
class Volvo{
      private boolean engineState;
      public String carModel;
      public Car(boolean e, String c){
            engineState = e;
            carModel = c;
      }
      public void startEngine(){
            engineState = true;
      }
      Public void pushGasPedal(){
            injectGasIntoCylinder();
      }
      public void shutOffEngine(){
            engineState = false;
      }
      private void injectGasIntoCylinder(){
            /* some implementation */
      }
}
```

Åtkomst

```
Class Main{
      public static void Main(String[] args){
            Car c1 = new Car(false, “V40”);
            Car c2 = new Car(false, “XC90”);
            c2.startEngine();
            c2.pushGasPedal();
            c2.shutOffEngine();

      }
}
```

Rätt

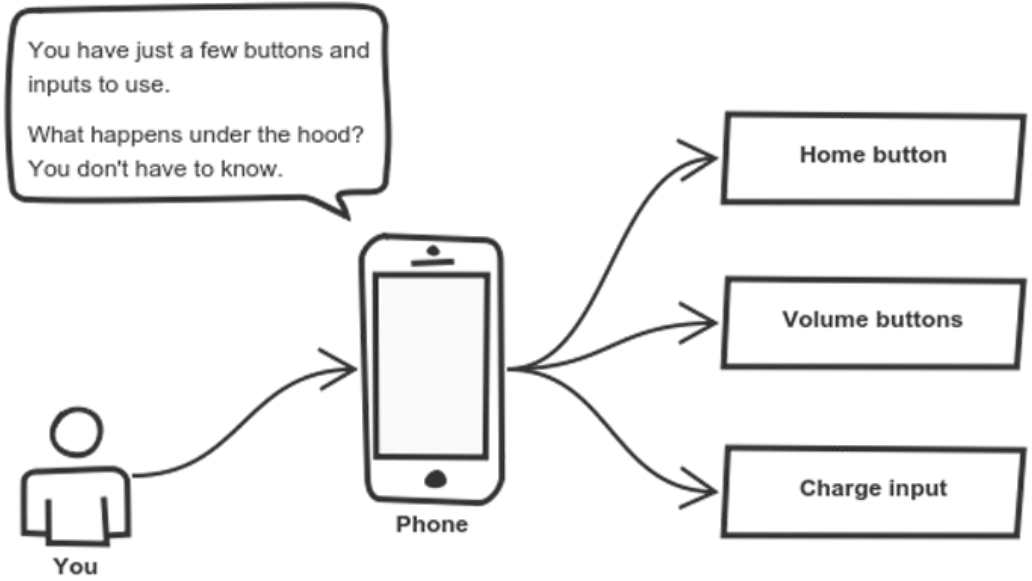
```
Class Main{
      public static void Main(String[] args){
            Car c1 = new Car(false, “V40”);
            Car c2 = new Car(false, “XC90”);
            c2.startEngine();
            c2.injectGasIntoCylinder();
            c2.shutOffEngine();

      }
}
```

Fel

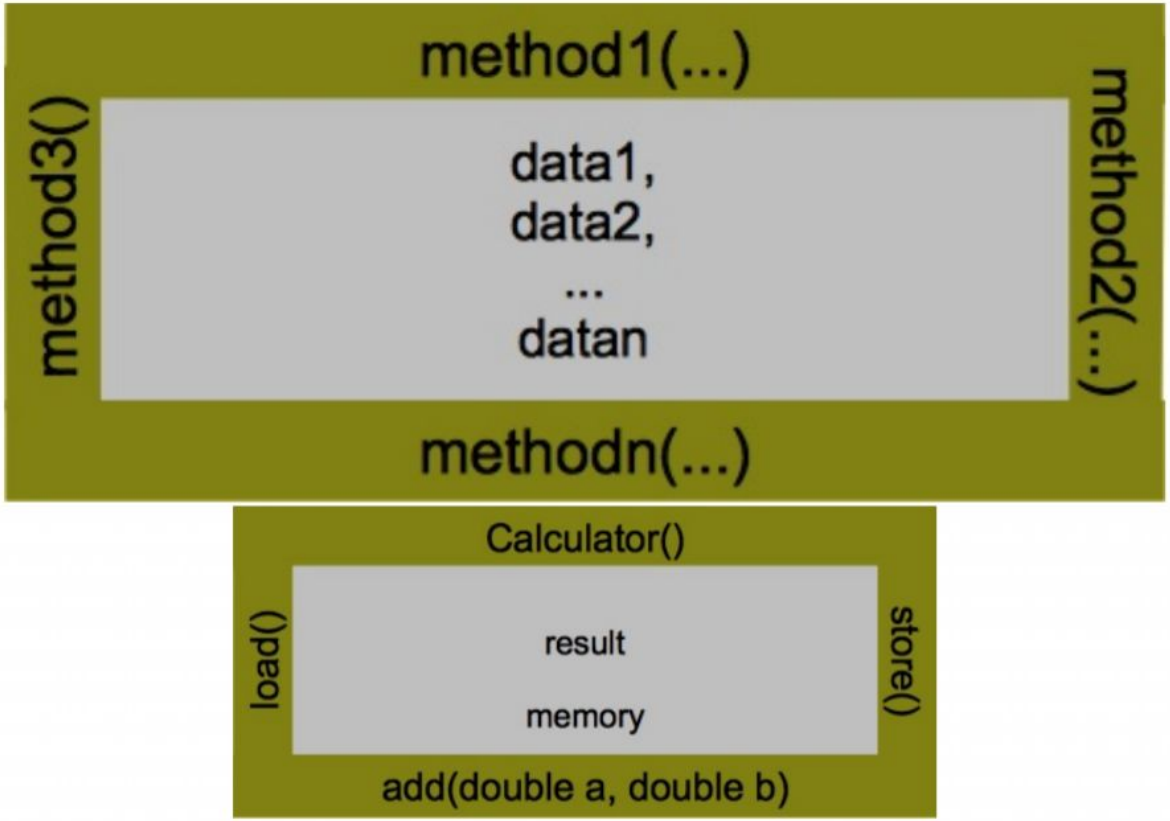
Inkapsling(Encapsulation)

Inkapsling (Encapsulation)



LINKÖPINGS UNIVERSITET

Inkapsling(Encapsulation)



LINKÖPINGS UNIVERSITET

Inkapsling (Encapsulation)

- Alla fält deklareras som `private`.
- Metoder kan vara `public, default, protected` eller `private`.
- Fält innehåller data som ska döljas från omvärlden.

Inkapsling(Encapsulation)

	Public	Private
Fält	Bryter mot inkapsling	Verkställer inkapsling
Metoder	Verkställer inkapsling	Hjälper egna metoder

Inklpsling(Encapsulation)

Syntax:

```
class MobilePhone{
      private int volume;
      private int chargeLevel;
      public MobilePhone(int volume){
            this.volume = volume;
      }
      public void home(){
            showHomeScreen(); // a private method
      }
      Public void volumeUp(){
            volume++;
      }
      public void volumeDown(){
            volume--;
      }
      private void startCharging(){
            readElctricity(); // a private method
      }
}
```

Inkapsling (Encapsulation)

- Man kan också använda `setter` och `getter` metoder för att tillåta andra jobba med inkapslade data.
- Varje privat variabel kan ha en publik getter.
- Varje privat variabel kan ha en publik setter.
- Namnet på en standard setter/getter börjar med ordet “set” eller “get” som följs av namnet på inkapslade variabeln.
- Exempel:

```
class Student{
    private String id;
    public void setId(String id){
        this.is = id;
    }
    public String getId(){
        return id;
    }
}
```

LINKÖPINGS UNIVERSITET

Inklpsling(Encapsulation)

Syntax:

```
class MobilePhone{
      private int volume;
      private int chargeLevel;
      public MobilePhone(int volume){
            setVolume(volume);
      }
      public void home(){
            showHomeScreen(); // a private method
      }
      public void setVolume(int volume){
            this.volume = volume;
      }
      public int getVolume(){
            return volume;
      }

      private void startCharging(){
            readElctricity(); // a private method
      }
}
```

LINKÖPINGS UNIVERSITET

2D-Arrays

2D-arrays

Antalet rader

Antalet kolumner

- Arrays kan ha två dimensioner.
- Såna arrays presenterar matriser.
- Exempelkod på hur man kan skapa 2D-arrays:
  - `int[][] a = new  int[3][4]; //12 block totalt`

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

LINKÖPINGS UNIVERSITET

2D-arrays

- Att skapa samma array på ett annat sätt:
  - `int[][] a = {{1, 2, 3, 4}, {5, 6, 7, 8}, {9, 10, 11, 12} }`

	Column 1	Column 2	Column 3	Column 4
Row 1	a[0][0]	a[0][1]	a[0][2]	a[0][3]
Row 2	a[1][0]	a[1][1]	a[1][2]	a[1][3]
Row 3	a[2][0]	a[2][1]	a[2][2]	a[2][3]

2D-arrays

- Det kan finnas olika antal kolumner per rad:
  - `int[][] a = {{1, 2,}, {5, 6, 7, 8}, {9, 10, 11}}`

2D-arrays

- Det kan finnas olika antal kolumner per rad:
  - `int[][] a = new int[3][];`
  - `a[0] = new int[2];`
  - `a[1] = new int[4];`
  - `a[2] = new int[3];`

Java Standard Bibliotek

Standard bibliotek

- `java.lang`
  - Innehåller fundamentala klasser.
  - Importeras automatiskt.
- `java.util`
  - Innehåller klasser som hjälper med att göra olika uppgifter.
- `java.io`
  - Innehåller klasser som används för att lagra data på och läsa data från hårddisken.

Wrapper klasser(java.lang)

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

Wrapper klasser

- Wrapper klasser presenterar primitiva datatyper.
- De kan användas i stället för primitiva datatyper.
- exempelkod:

```
public class Main {
  public static void main(String[] args) {
    Integer myInt = 5;
    Double myDouble = 5.99;
    Character myChar = 'A';
    System.out.println(myInt);
    System.out.println(myDouble);
    System.out.println(myChar);
  }

}
```

Datatyp-omvandling

Från String till primitiv data:

Vi använder wrapper-klassen till datatypen som vi vill omvandla till. T.ex. vi använder Integer om vi vill omvandla “3” till 3. Integer har två statiska metoder som vi kan använda:

parseInt: tar emot en sträng och returnerar en int.
valueOf: tar emot en sträng och returnerar en Integer.

Från primitiv data till String:

Vi använder klassen String. T.ex. vi använder String om vi vill omvandla 3 till “3”. Klassen String har en statisk metod som tar emot en integer och returnerar en sträng.

valueOf: tar emot en Integer och returnerar en String.

Kan vi omvandla primitiv data till String på ett annat sätt?

Thanks for listening!

LINKÖPINGS UNIVERSITET