TDDC77
Objektorienterad Programmering

Föreläsning 6

Sahand Sadjadee
IDA, Linköpings Universitet
Hösttermin 2020

Outline

- Överlagring (Overloading)
- Åtkomt (Access)
- Inkapsling (Encapsulation)
- Abstrakta datatyper (Abstract Datatypes)

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 konkret metod deklareras med hjälp av fyra komponenter
  - Returtyp eller void
  - Metodens namn (identifierare) som ska börja med liten bokstav
  - Parameterlista med noll eller fler datatyper och identifierare
  - Metodens kropp inom måsvingar
- Metodens signatur bestäms av metodens namns samt datatyper och ordning på parameterlistan.

Överlagring

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

Task 1- 5 minuter

Skriv en klass som heter `Math`. Klassen ska innehålla två metoder som båda heter `add` och kan användas för att addera heltal. Första metoden ska ta emot två heltal och andra ska ta emot tre heltal.

Åtkomst(Access)

Åtkomst

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

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

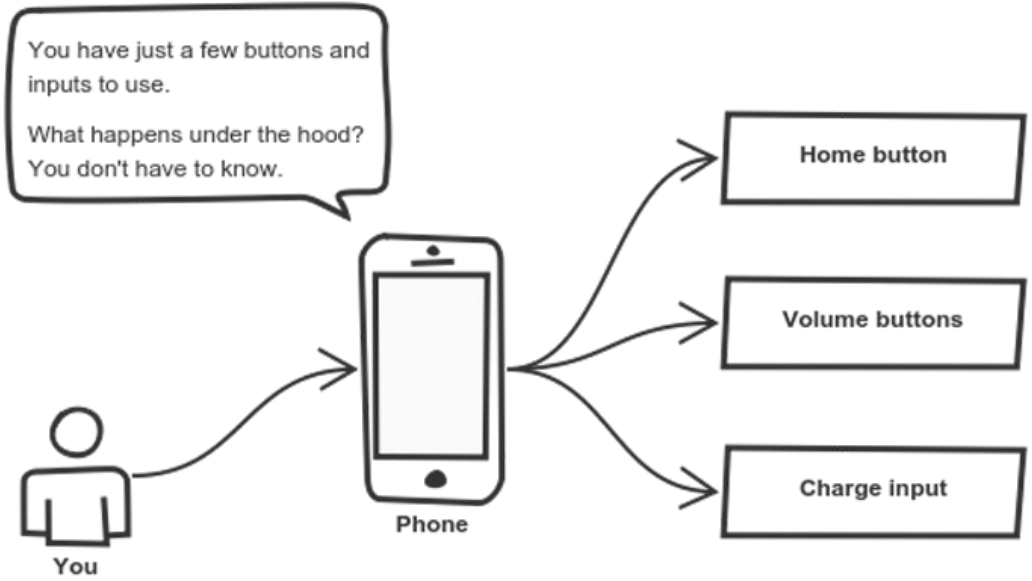
```
 Class Main{
       public static void Main(String[] args){
             Car c1 = new Car(false, "V40");
             Car c2 = new Car(false, "XC90");
             c2.startEngine();
             c2.injectGasIntoCylinder();
             c2.shutOffEngine();

       }
 }
```

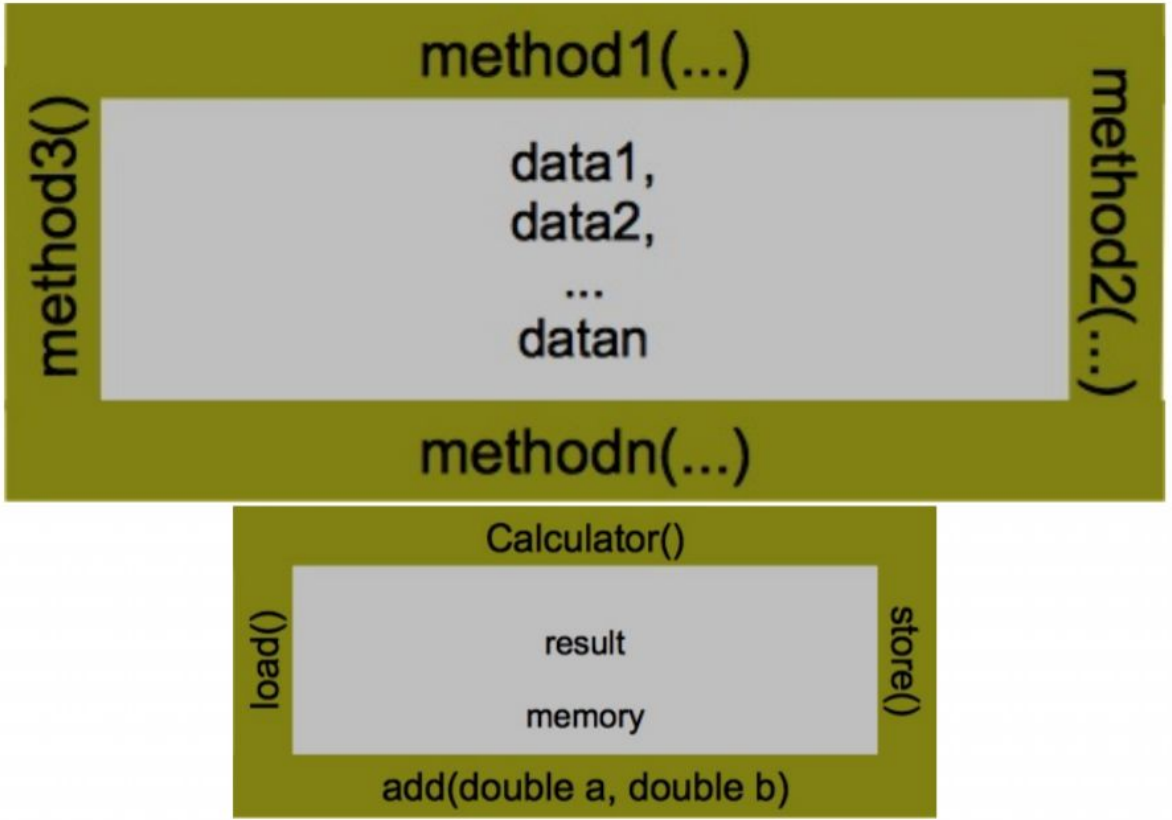
Fel

Inkapsling(Encapsulation)

Inkapsling (Encapsulation)



Inkapsling(Encapsulation)



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 låta andra jobba med inkapslade data.
- Varje privat variabel kan ha en eller flera publika getters.
- Varje privat variabel kan ha en eller flera publika setters.
- 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;
     }

}
```

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
      }
}
```

Abstrakta Datatyper(ADT)

ADT

- Ibland vill man skapa nya datatyper med särskilda operationer som kan utföras på data.
- En abstrakt datatyp definieras av de operationerna som kan utföras på data och inte hur data lagras.

Exempel

Definition på en stapel(Stack):

- push()
- pop()
- length()

Definition på en pixel:

- getX()
- getY()
- getColor()

ADT

- För att skapa en ny abstrakt datatyp ska en ny klass skapas med fält, data som lagras, och metoder, operationer som ska utföras på data.
- Abstrakta datatyper döljer hur data lagras i bakgrunden genom att använda `private` nyckelordet för variablerna(fält).

Task 2 - 5 minuter

Skapa en klass, `Point`, som kan innehålla en pixels position, (X,Y), på skärmen. Det ska gå att addera och jämföra två separata pixlar.

Java har egen Point!

https://docs.oracle.com/javase/7/docs/api/java/awt/Point.html

- I verklighet ska det kollas om den datatypen som man vill skapa redan finns i standard biblioteket eller inte. Om den finns och gör vad man vill den göra då ska man skippa skapa egen datatyp.
- Man kan använda andra ADT, egna eller från standard biblioteket, för att skapa nya ADT.

Task 3 - 5 minuter

Skapa en klass, Pane, som kan användas för att definiera en ruta på skärmen. Det ska gå att jämföra två separata rutors yta.

Viktiga punkter

- Att använda metoder för att manipulera de abstrakta datatyper, i stället för att direkt komma åt de interna variablerna.
- Kan ändra hur datatypen är implementerad utan att behöva ändra den kod som använder de abstrakta datatyper.

Thanks for listening!

LINKÖPINGS UNIVERSITET