Lektionstillfälle 2
Essensen av OO i Java

Avancerad Javaprogrammering

Utbildare: Mikael Olsson

Lektionstillfällets mål och metod

Mål med lektionen:

- Essensen av objektorientering i Java

Lektionens arbetsmetod/er:

- Micke föreläser
- Eleverna gör övningsuppgift 2

Kort summering av föregående lektion

- Objektorientering är att modellera kod efter mänskligt tänkande
- Objekt (klasser) kan oftast beskrivas med substantiv
- Ett objekts attribut uttrycks med instansvariabler
- ”A är B” utrycks genom **att A ärver B**
- ”A har B” uttrycks genom att **A har instansvariabler av typ B**

Metoders synlighet 1

- En subklass har tillgång till följande av superklassens metoder:
  - Public
  - Protected

- INTE private-metoder (observera röd understrykning)

```
class Huvud {
    private Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(Öga h, Öga v){
        högerÖga = h;
        vänsterÖga = v;
    }
    public String getEyeColor(){
        return vänsterÖga.getColor();
    }
    private void swapEyes(){
        högerÖga = vänsterÖga;
        vänsterÖga = högerÖga;
    }
}

class DockHuvud extends Huvud {
    public DockHuvud(Öga porslinsÖga){
        super(porslinsÖga,porslinsÖga);
    }
    public void givtEyeColorCompliment(){
       System.out.println("You have pretty "
               +getEyeColor()+" eyes.");
    }
    public void partyTrick(){
       swapEyes();
    }
}
```

Metoders synlighet 2

Vad händer om det finns metoder med samma namn i både super- och subklass?

- Fullt tillåtet
- Vanligt
- Subklassens metod överskuggar superklassens
- Använd annotationen @Override
  - För tydlighetens skull
  - Slipper fel vid stavfel

```
class Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(){}

    public void printMe(){
        System.out.println("Jag är ett huvud");
    }
}

class DockHuvud extends Huvud {
    public DockHuvud(){}

    @Override
    public void printMe(){
        System.out.println("Jag är ett dockhuvud");
    }
}
```

Variablers synlighet 1

En subklass tillgång till följande av superklassens instansvariabler:

- Public
- Protected

- INTE private-variabler (observera röd understrykning)

```
class Huvud {
    private Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(Öga h, Öga v){
        högerÖga = h;
        vänsterÖga = v;
    }
}

class DockHuvud extends Huvud {
    public DockHuvud(Öga porslinsÖga){
        super(porslinsÖga,porslinsÖga);
    }
    public Öga getVänsterÖga(){
       return vänsterÖga;
    }
    public Öga getHögerÖga(){
       return högerÖga;
    }
}
```

Variablers synlighet 2

Vad händer om det finns variabler med samma namn i både super- och subklass?

- Fullt tillåtet, men rörigt
- Separata variabler, trots samma namn
- Subklassens variabler överskuggar superklassens
- Dockhuvud.högerÖga ger det högerÖga som är definierat i Dockhuvud och inte det som är definierat i Huvud.
- Om du har ett Dockhuvud är Huvuds högeröga gömd och du kommer inte åt den
- Stor risk för fel, försök undvika detta!

```
class Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(){}
}

class DockHuvud extends Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;

    public DockHuvud(Öga porslinsÖga){
        super.högerÖga = porslinsÖga;
        super.vänsterÖga = porslinsÖga;
        this.högerÖga = porslinsÖga;
        this.vänsterÖga = porslinsÖga;
    }
}
```

Instansiering

- När du skriver `new XXX()` instansieras plats i minnet för ett objekt av typ XXX.
- En subklass innehåller allt som dess superklass gör och lite till.
- Om X är en superklass till XX kan du skapa en referensvariabel av typ X som refererar till XX.
- Du kan INTE skapa en referensvariabel av typ XX som refererar till X, då förväntar vi oss att alla XX:s attribut ska finnas, men vi har bara tillgång till Xs attribut.

```
class Huvud {
    private Öga vänsterÖga;
    private Öga högerÖöga;
    public Huvud(){}
}

class KattHuvud extends Huvud {
    private Nos nos;
    public KattHuvud(){}

    public static void main(String[] args){
        Huvud h1 = new Huvud();  //ok
        Huvud h2 = new KattHuvud();  //ok
        KattHuvud h3 = new KattHuvud();  //ok
        KattHuvud h3 = new Huvud();   //NEJ!
    }
}
```

Variablers synlighet 3

Vad skrivs ut?

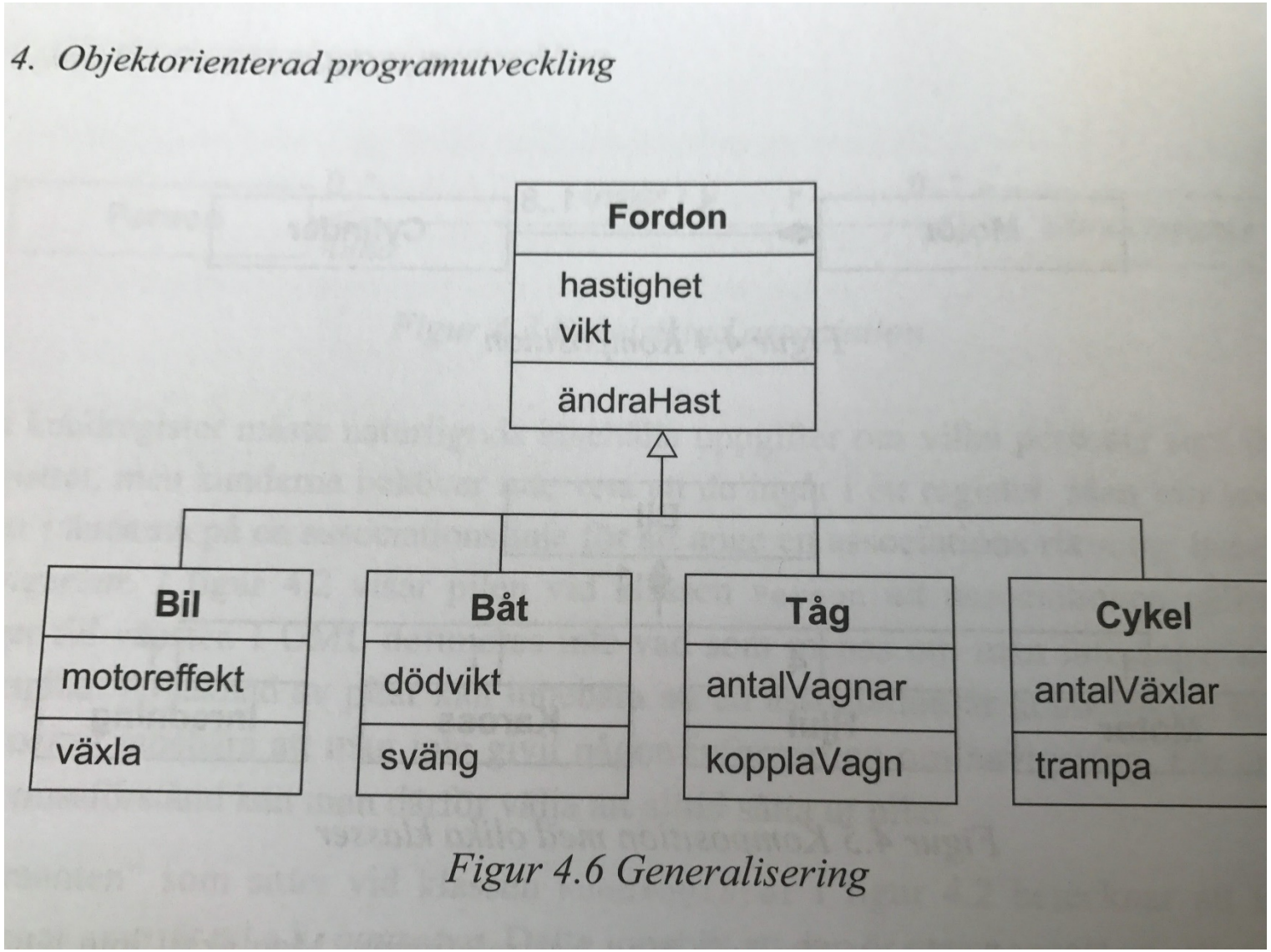
- h1.printÖga() => blått
- h2.printÖga() => grönt
- h3.printÖga() => blått

- Referensens typ styr vilken variabel som anropas

```
class Huvud {
    protected Öga öga;
    public Huvud(){
        öga = new Öga("blått");
    }
}

class DockHuvud extends Huvud {
    protected Öga öga;
    public DockHuvud(){
        this.öga = new Öga("grönt");
    }

    public static void main(String[] args) {
        Huvud h1 = new Huvud();
        DockHuvud h2 = new DockHuvud();
        Huvud h3 = new DockHuvud();

        System.out.println(h1.öga.färg);
        System.out.println(h2.öga.färg);
        System.out.println(h3.öga.färg);
    }
}
```

Övningsuppgift 2a

- Konstruera klasserna Fordon och Tåg från klassdiagrammet:
- Följ best practices för inkapsling och synlighet



Övningsuppgift 2a, lösning

```java
public class Tåg extends Fordon {
    private int antalVagnar;

    public Tåg(int hastighet, int vikt, int antalVagnar) {
        super(hastighet, vikt);
        this.antalVagnar = antalVagnar;
    }

    public void kopplaVagn(){}
}
```

```java
public class Fordon{

    private int hastighet;
    private int vikt;

    Fordon(){}

    Fordon(int hastighet, int vikt){
        this.hastighet = hastighet;
        this.vikt = vikt;
    }

    public int getHastighet(){
        return hastighet;
    }

    public int getVikt(){
        return vikt;
    }

    public void ändraHastighet(int nyHastighet){
        hastighet = nyHastighet;
    }
```

Abstrakt superklass

```
abstract class Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(){}
    abstract void printMe();
}
```

- Anges med `abstract`
- Anger att en klass inte ska gå att skapa instanser av
- Klassen går bra att ärva
- Används ofta för abstrakta begrepp
- T.ex kan klassen Däggdjur vara abstrakt, och implementeras av de icke-abstrakta klasserna ”Katt”, ”Hund” och ”Orm”

Abstrakt metod

```
abstract class Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(){}
    abstract void printMe();
}
```

- Anges med `abstract`
- Kan deklareras i abstrakta klasser
- En placeholder
- Icke-abstrakta subklasser måste implementera metoden

Final

- Ett skyddat ord som anger att en klass ALDRIG ska gå att ärva ifrån

```
final class DockHuvud extends Huvud {
    public DockHuvud(Öga porslinsÖga){
        super(porslinsÖga,porslinsÖga);
    }
    public void giveEyeColorCompliment(){
       System.out.println("You have pretty "
               +getEyeColor()+" eyes.");
    }
}
```

Övningsuppgift 2b

- Fixa din lösning till 2a så att Fordon blir en abstrakt klass.
- Lägg till den abstrakta metoden printMe() i Fordon.
- Implementera printMe() i Tåg. PrintMe ska skriva ut vikt, hastighet och antal vagnar.

Övningsuppgift 2b, lösning

```java
public class Tåg extends Fordon implements Printable{
    private int antalVagnar;

    public Tåg(int hastighet, int vikt, int antalVagnar) {
        super(hastighet, vikt);
        this.antalVagnar = antalVagnar;
    }

    public void kopplaVagn(){}

    public void printMe(){
        System.out.println("Tåg med hastighet "
                +getHastighet()+ " km/h, vikt "
                + getVikt() +" kg och antalVagnar: "
                + antalVagnar);
    }
}
```

```java
abstract class Fordon{

    private int hastighet;
    private int vikt;

    Fordon(){}

    Fordon(int hastighet, int vikt){
        this.hastighet = hastighet;
        this.vikt = vikt;
    }

    public int getHastighet(){
        return hastighet;
    }

    public int getVikt(){
        return vikt;
    }

    public void ändraHastighet(int nyHastighet){
        hastighet = nyHastighet;
    }

    //Nödvändig för att det ska gå att skriva
    //ut en subklass via Fordon
    abstract public void printMe();
}
```

Objektorienteringens 4 principer

1. Inkapsling
   - Hemlig inre implementation
   - Tydliga kontaktytor mellan objekt
2. Abstraktion
   - Kodmodellerna bygger på verkligheten
3. Arv
   - Uttrycker generalisering/specialisering
4. Polymorfism
   - Ett anrop till samma metod leder till anrop på olika instansmetoder beroende på objektets typ

Polymorfism

Huvud a = new DockHuvud();

Huvud b = new HästHuvud();

a.printMe() => ”Dockhuvud”

b.printMe() => ”Hästhuvud”

- Objektets typ styr vilken metod anropas
- OBS, tvärt om mot variabler (där styr referensens typ)

```
abstract class Huvud {
    protected Öga högerÖga;
    protected Öga vänsterÖga;
    public Huvud(){}
    abstract void printMe();
}

class DockHuvud extends Huvud {
    public DockHuvud(){}

    public void printMe(){
        System.out.println("Dockhuvud");
    }
}

class HästHuvud extends Huvud {
    public HästHuvud(){}

    public void printMe(){
        System.out.println("Hästhuvud");
    }
}
```

Dynamisk bindning

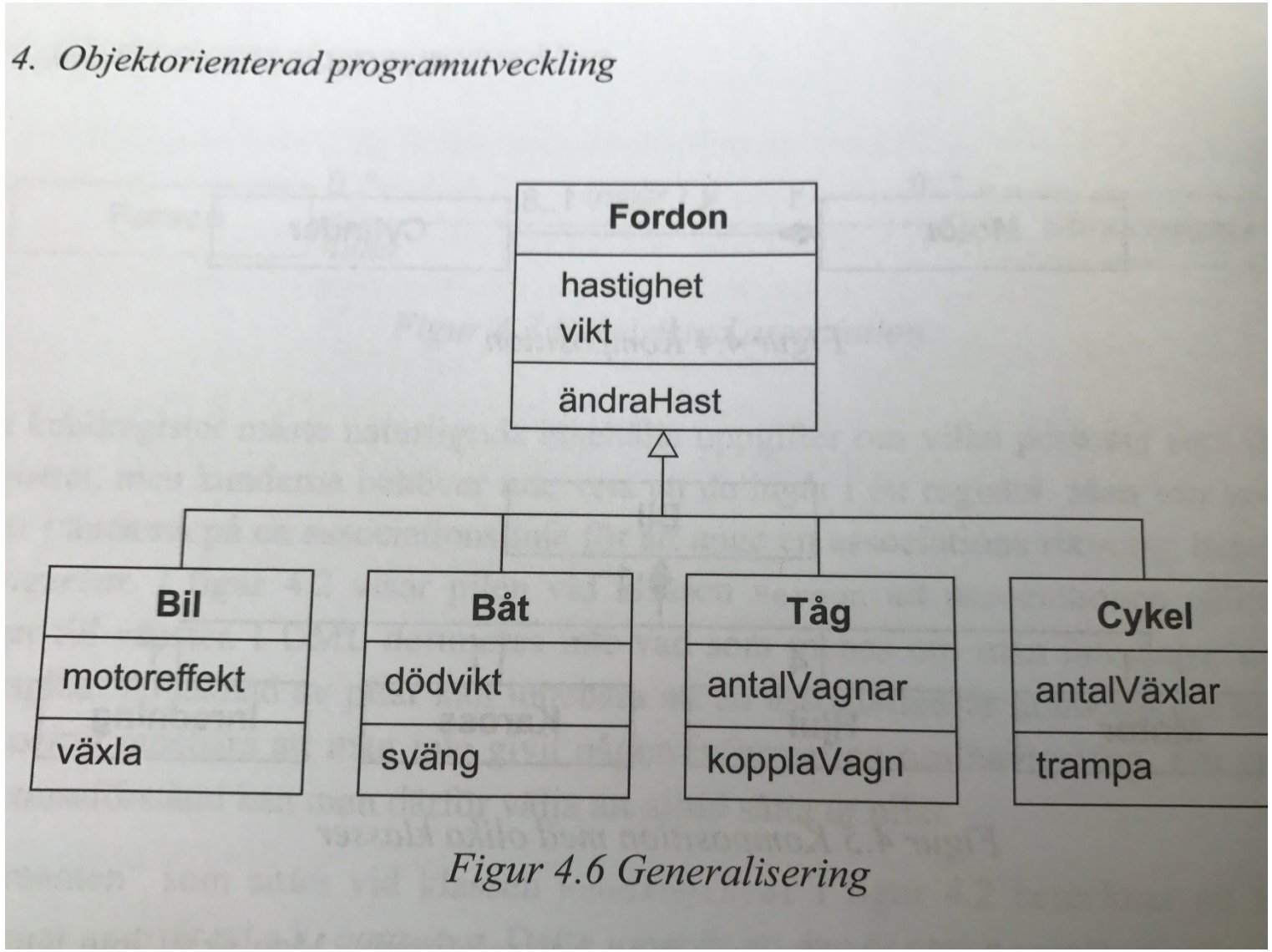
- När en instansmetod anropas under exekvering kollar Javamotorn om det finns en passande metod i den klass som objektet i fråga tillhör.
  - Om ja, använd den metoden
  - Om nej, fortsätt ett snäpp uppåt i arvshierarkin och leta där
  - Upprepa tills lämplig metod hittas, annars generera fel

Konstruktorer i arvshierarkier

- När ett objekt O instansieras skapas först ett objekt av Os översta superklass. Sedan klassen under, sedan klassen under, tills vi har hittat klassen O och skapat en instans av den.
- Tänk ryska dockor, vi jobbar inifrån och ut.
- Super() anropar min superklass konstruktor.

Övningsuppgift 2c, 2d

- Utöka 2b genom att skapa klasserna Bil, Båt och Cykel enl. Figur 4.6.
- Låt metoden printMe() skriva ut vikt, hastighet och ytterligare egenskap för varje subklass.
- Lägg till ett program som skapar upp ett fordon av varje sort. Lägg till en metod som skriver ut data om de fordon du skapat. Anropa utskriftsmetoden från ditt huvudprogram.



Figur 4.6 Generalisering

Lösning uppgift 2c, 2d

```java
public class Cykel extends Fordon {
    private int antalVäxlar;

    public Cykel(int hastighet, int vikt, int antalVäxlar)
        super(hastighet, vikt);
        this.antalVäxlar = antalVäxlar;
    }

    public void trampa(){}

    public void printMe(){
        System.out.println("Cykel med hastighet "
                +getHastighet()+ " km/h, vikt "
                + getVikt() +" kg och antalVäxlar: "
                + antalVäxlar);
    }
}
```

```java
public void printFordon(Fordon fordon){
    fordon.printMe();
}

Övningsuppgift2(){
    Tåg tåg = new Tåg(180, 300000, 5);
    Bil bil = new Bil(200, 1000, 70);
    Båt båt = new Båt(50, 300, 50);
    Cykel cykel = new Cykel(20, 5, 10);

    printFordon(bil);
    printFordon(båt);
    printFordon(tåg);
    printFordon(cykel);
}
```

Interface 1

- Men om vi vill ärva flera olika klasser (som inte ärver varandra)?
  - Inte tillåtet i Java
- Istället finns konstruktionen ”Interface” = gränssnitt
- Som en abstrakt klass där alla metoder är abstrakta.
- Används för att beskriva vilka gemensamma egenskaper en grupp klasser har.
- Tänk ”förpackning”, kravlista, handle eller mall
- En klass får implementera hur många interface som helst.
- En klass implementerar ett interface genom att använda ”`implements`” i sin deklaration och implementera de metoder som interfacet påbjuder.

Interface 2

- En klass implementerar ett interface genom att implementera de metoder som interfacet påbjuder.
- Att en klass implementerar ett interface beskrivs med `implements`
- Om Hästhuvud implementerar Friendly MÅSTE metoden makeFriendlySound() vara implementerad i Hästhuvud

```
interface Friendly{
    void makeFriendlySound();
}

class HästHuvud extends Huvud implements Friendly{
    protected Öga öga;
    public HästHuvud(){
       this.öga = new Öga("grönt");
    }
    public void makeFriendlySound(){
        System.out.println("hmmmmmmmmmm");
    }
}

class KattHuvud extends Huvud implements Friendly{
    protected Öga öga;
    public KattHuvud(){
       this.öga = new Öga("grönt");
    }
    public void makeFriendlySound(){
        System.out.println("purrrrrrrrrrrrrr");
    }
}
```

Interface liknar klasser:

- Får innehålla godtycklig mängd metoder
- Filnamnen slutar på **.java**
- Ligger i paket i filstrukturen tillsammans med klasser

Interface är olika klasser:

- Kan inte instantieras
- Ingen konstruktor
- Alla metoder är abstrakta
- Har mycket sällan instansvariabler
  - De som finns måste vara final static
- Kan inte ärvas av en klass, bara implementeras
- Ett interface kan ärva multipla interface
  - Då används ”extends”

Om interface

- Interface är implicit abstrakta.
  - Därför behöver aldrig ordet abstract användas vid klassdeklaration
- Interfacens metoder är implicit abstrakta.
  - Därför behöver aldrig ordet abstract användas vid metoddeklaration
- Interface är implicit publika

Interface 3

- Det är tillåtet att deklarera referensvariabler som refererar till interface.
- Interface-variabeln kan INTE använda metoder hos objektet som inte är definierade i interfacet (se eatCatFood).

```
interface Friendly{
    void makeFriendlySound();
}

class KattHuvud extends Huvud implements Friendly{
    protected Öga öga;
    public KattHuvud(){
       this.öga = new Öga("grönt");
    }
    //included in Friendly interface
    public void makeFriendlySound(){
        System.out.println("purrrrrrrrrrrrrr");
    }
    //not included in Friendly
    public void eatCatFood(){
        System.out.println("mums mums");
    }

    public static void main(String[] args){
        Friendly f = new KattHuvud();
        f.makeFriendlySound();
        f.eatCatFood();
    }
}
```

Interface i Java

- Gå in på https://docs.oracle.com/javase/8/docs/api/
- Titta i rutan ”All Classes”
- Alla ”klasser” som står med kursiv stil är egentligen interfaces

Övningsuppgift 2e

- Ändra din lösning på 2d. Lägg till interfacet Printable och deklarera metoden printMe där.
- Låt dina fordonsklasser implementera interfacet Printable.
- Gör en ny utskriftsmetod I huvudklassen som skriver ut fordonens data genom att anropa printMe via Printable-interfacet. Lägg till ett anrop till denna nya metod från ditt huvudprogram.

Övningsuppgift 2e, lösning

```
public interface Printable {
    void printMe();
}
```

```
public class Cykel extends Fordon implements Printable{
    private int antalVäxlar;

    public Cykel(int hastighet, int vikt, int antalVäxlar) {
        super(hastighet, vikt);
        this.antalVäxlar = antalVäxlar;
    }

    public void trampa(){}

    public void printMe(){
        System.out.println("Cykel med hastighet "
                +getHastighet()+ " km/h, vikt "
                + getVikt() +" kg och antalVäxlar: "
                + antalVäxlar);
    }
}
```

```
public void printFordon(Fordon fordon){
    fordon.printMe();
}

//tillhör Uppgift 2e
public void printViaInterface(Printable whatever){
    whatever.printMe();
}

Övningsuppgift2(){
    Tåg tåg = new Tåg(180, 300000, 5);
    Bil bil = new Bil(200, 1000, 70);
    Båt båt = new Båt(50, 300, 50);
    Cykel cykel = new Cykel(20, 5, 10);

    printFordon(bil);
    printFordon(båt);
    printFordon(tåg);
    printFordon(cykel);

    System.out.println();

    //tillhör Uppgift 2e
    printViaInterface(bil);
    printViaInterface(båt);
    printViaInterface(tåg);
    printViaInterface(cykel);

}
```

Övningsuppgift 2f

- Lägg till ett ytterligare interface “Hjulburen” som har metoden getAntalHjul()
- Låt relevanta klasser implementera interfacet (lägg till instansvariabler vid behov)
- Skapa upp några “Hjulburen” av olika typer och skriv ut antalet hjul för varje fordon.

Sammanfattning

- Du kan referera till ett objekt via objektets superklasser
- Du kan också referera till ett objekt via de interface objektet implementerar
- Polymorfism innebär att 2 anrop till samma metod i en klass kan resultera i anrop till olika metoder beroende på vilken typ de objekt som refereras har.
- Dynamisk bindning innebär att när en metod hos en subklass anropas så kastas anropet uppåt i arvshierarkin tills en korrekt metod hittas.
- Interface är Javas svar på multipla arv