EDA016 Programmeringsteknik för D

Läsvecka 7: Arv

Björn Regnell

Datavetenskap, LTH

Lp1-2, HT 2015

7 Arv

- Att göra denna vecka
- Grundläggande terminologi för arv
- Klassificering och abstrakta klasser
- Operatorn `instanceof`
- Klassen `Object`
- Attributarv, `protected` och `super`
- Implementera `equals` och `toString`

Att göra i Vecka 7: Förstå arv. Repetera inför kontrollskrivning.

1. Läs följande kapitel i kursboken:
   9.1, 9.3, 9.7-9.9, 11.2, 12.6
   Begrepp: arv, subklass, superklass, **`extends`**, **`abstract`**, operatorn **`instanceof`**, `Object`, **`super`**, `equals`,
2. Gör övning 7: registrering
3. Träffas i samarbetsgrupper och hjälp varandra
4. Gör Lab 6: Implementera Turtle och ColorTurtle.
5. Plugga inför **obligatorisk** kontrollskrivning.

Vad är arv? Motivering och terminologi

- Med hjälp av **arv** mellan klasser kan man göra så att en klass **ärver** (”får med sig”) innehållet i en *annan* klass.
- Varför vill man det?
  1. Dela upp ansvar mellan klasser och bryta ut gemensamma delar så att man slipper duplicerad kod.
  2. Skapa en klassificering av objekt utifrån relationen **X är en Y**.
     Exempel 1: En gurka är en grönsak. En tomat är en grönsak.
     Exempel 2: En cykel är ett fordon. En bil är ett fordon.
- Nyckelordet `extends` används för att ange arv i Java.
  Exempel: `class TalkingRobot extends Robot`
- Klassen som ärver (utökar) kallas **subklass**
- Klassen som blir utökad kallas **superklass** (även *basklass*)
- Läs mer om arv (eng. *inheritance*) här:
  https://sv.wikipedia.org/wiki/Arv_%28programmering%29

Exempel: Robot

Krav:
Det ska finnas två sorters robotar: `Robot` och `TalkingRobot`.
Båda kan arbeta, men bara den senare kan prata.

En möjlig **design** om vi *inte* använder arv:

`Robot`

```
/** Arbeta */
public void work();
```

`TalkingRobot`

```
/** Arbeta */
public void work();

/** Prata */
public void talk();
```

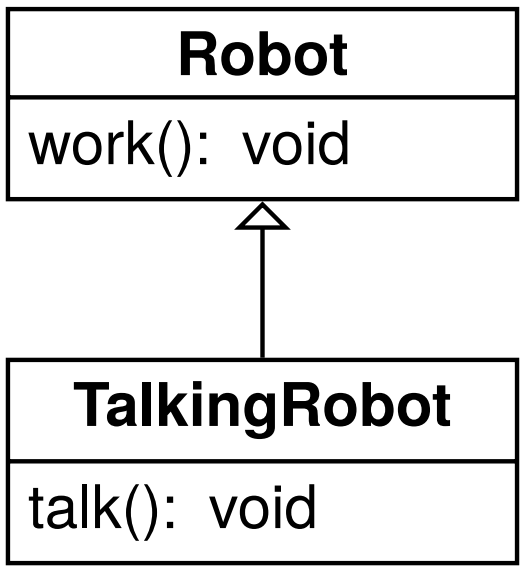
Utökning med copy-paste ger duplicerad kod

```
1  package week07.robotduplicate;
2
3  class Robot {
4      public void work() {
5          System.out.println("Robot is workign.");
6      }
7  }
8
9  class TalkingRobot {
10     public void work() {                          // Code duplication with copy-paste.
11         System.out.println("Robot is workign."); // Need to fix misspelling in 2 places :(
12     }
13
14     public void talk() {
15         System.out.println("I shall not harm humans.");
16     }
17 }
18
19 public class RobotNoInheritance {
20     public static void main(String[] args) {
21         Robot wallE = new Robot();
22         wallE.work();
23         TalkingRobot c3po = new TalkingRobot();
24         c3po.talk();
25         c3po.work();
26     }
27 }
```

Utökning med arv – vi slipper duplicera kod

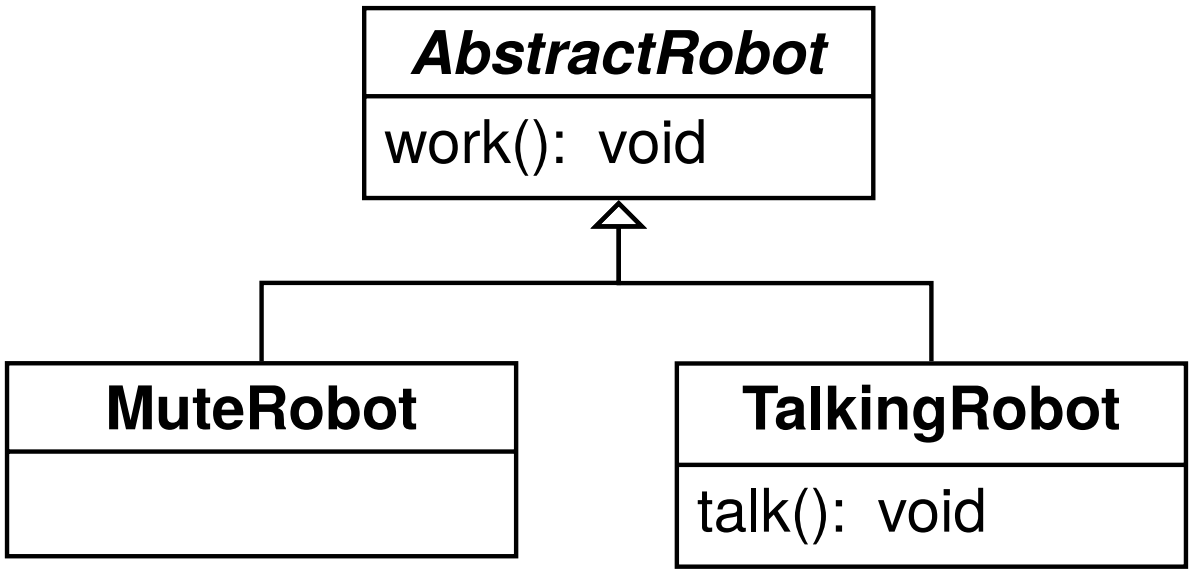
```
1  package week07.robotinherit;
2
3  class Robot {
4      public void work() {
5          System.out.println("Robot is working.");
6      }
7  }
8
9  class TalkingRobot extends Robot {  // TalkingRobot inherits the contents of Robot
10     public void talk() {
11         System.out.println("I shall not harm humans.");
12     }
13 }
14
15 public class RobotInheritance {
16     public static void main(String[] args) {
17         Robot wallE = new Robot();
18         wallE.work();
19         TalkingRobot c3po = new TalkingRobot();
20         c3po.talk();
21         c3po.work();
22     }
23 }
```

Illustrera arvsrelationer i diagram

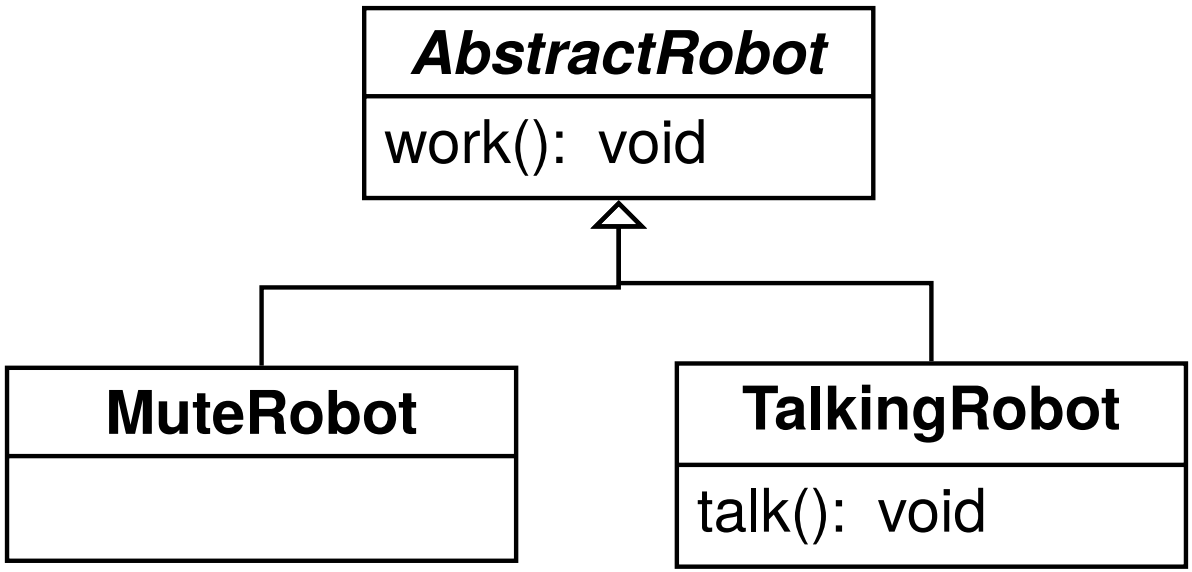


Class Diagram, Unified Modelling Language (UML), Ankboken sid 147.

Klassificering av robotar



Klassificering av robotar



- **Abstrakta klasser** används som bas för klassificering
- Deklaration i Java: `abstract class ClassName`
- Försök att instantiera abstrakta klasser ger kompileringsfel.
- `MuteRobot` är en **konkret klass** som kan instantieras och ärver `work()`-metoden från `AbstractRobot`

Exempel: klassificeringsmodell i Java

```java
package week07.robotclassification;

public abstract class AbstractRobot {
    public void work() {
        System.out.println("Robot is working.");
    }
}
```

```java
package week07.robotclassification;

public class TalkingRobot extends AbstractRobot {
    public void talk() {
        System.out.println("I shall not harm humans.");
    }
}
```

```java
package week07.robotclassification;

public class MuteRobot extends AbstractRobot {
    //empty class body; nothing added to AbsractRobot
}
```

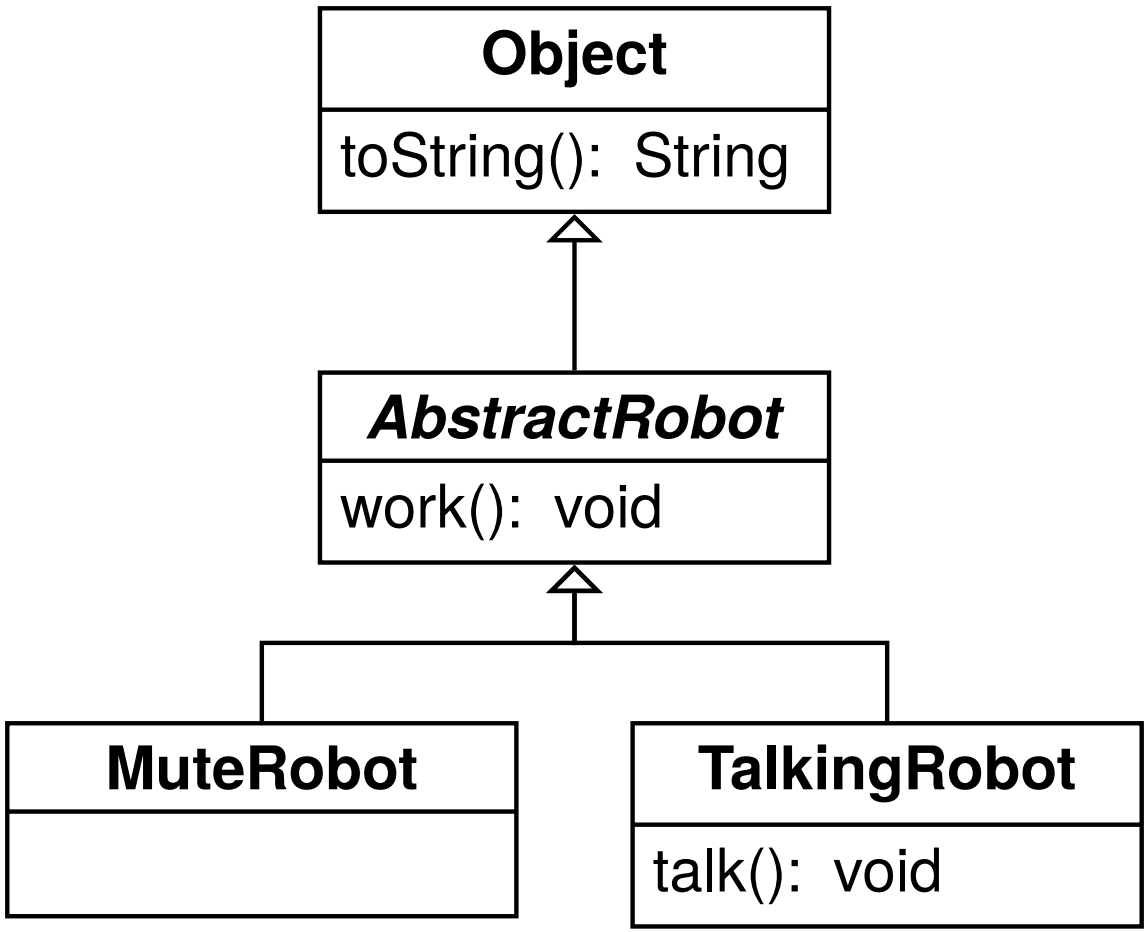
Exempel: TestRobots

```
1  package week07.robotclassification;
2
3  public class TestRobots {
4      public static void main(String[] args) {
5          MuteRobot wallE = new MuteRobot();
6          TalkingRobot c3po = new TalkingRobot();
7          // AbstractRobot r = new AbstractRobot(); //compile error: "Cannot instantiate..."
8          wallE.work();
9          // wallE.talk(); //compile error: "The method talk() is undefined..."
10         c3po.talk();
11         c3po.work();
12     }
13 }
```

Operatorn instanceof

```
1  package week07.robotclassification;
2  public class TestInstanceOf {
3      public static void main(String[] args) {
4          MuteRobot wallE = new MuteRobot();
5          TalkingRobot c3po = new TalkingRobot();
6          if (wallE instanceof MuteRobot) {
7              System.out.println("Wall-E kan inte prata");
8          }
9  //      if (wallE instanceof TalkingRobot) { // compile error
10 //          System.out.println("Wall-E kan inte prata");
11 //      }
12         if (c3po instanceof TalkingRobot) {
13             System.out.println("C3PO kan prata");
14         }
15         if (wallE instanceof AbstractRobot){
16             System.out.println("Wall-E kan arbeta");
17         }
18         if (c3po instanceof AbstractRobot){
19             System.out.println("C3PO kan arbeta");
20         }
21         if ((wallE instanceof Object) && (c3po instanceof Object)){
22             System.out.println("Wall-E och C3PO är javaobjekt.");
23         }
24         TalkingRobot r = null;
25         if (r instanceof TalkingRobot){ // Is this true or false??
26                System.out.println(r + " instanceof TalkingRobot == true");
27         } else System.out.println(r + " instanceof TalkingRobot == false");
28     }
29 }
```

Alla objekt är implicita instanser av klassen `Object`



Attributarv, protected och super

```java
package week07.superconstructor;

public class Robot {
    protected String name;  // protected: visible in subclasses

    public Robot(String name) {
        this.name = name;
    }

    public void work() {
        System.out.println(name + " is working.");
    }
}
```

```java
package week07.superconstructor;

public class TalkingRobot extends Robot {
    public TalkingRobot(String name) {
        super(name);  // call superclass constructor
    }

    public void talk() {
        System.out.println(name + " shall not harm humans."); // name visible in subclass
    }
}
```

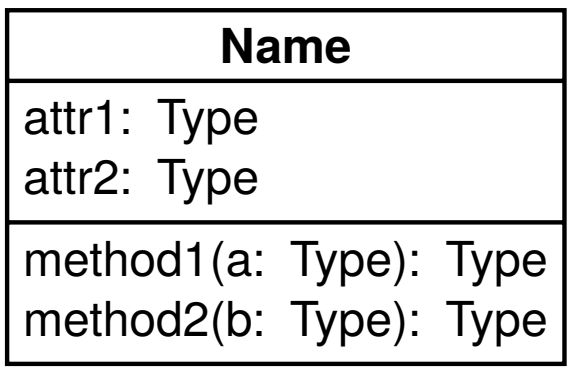
Test av protected och super

```
1  package week07.superconstructor;
2
3  public class TestSuperConstructor {
4      public static void main(String[] args) {
5          Robot wallE = new Robot("Wall*E");
6          TalkingRobot c3po = new TalkingRobot("C-3PO");
7          wallE.work();
8          c3po.talk();
9          c3po.work();
10     }
11 }
```

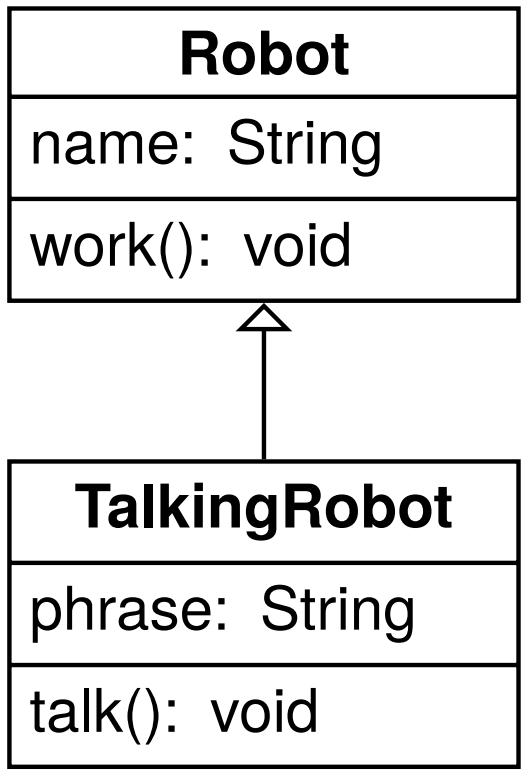
```
Wall*E is working.
C-3PO shall not harm humans.
C-3PO is working.
```

Attribut i UML-diagram

En klass i UML kan ha 3 delar:



Ibland utelämnar man typerna.
Se exempel i ankboken sid. 162.

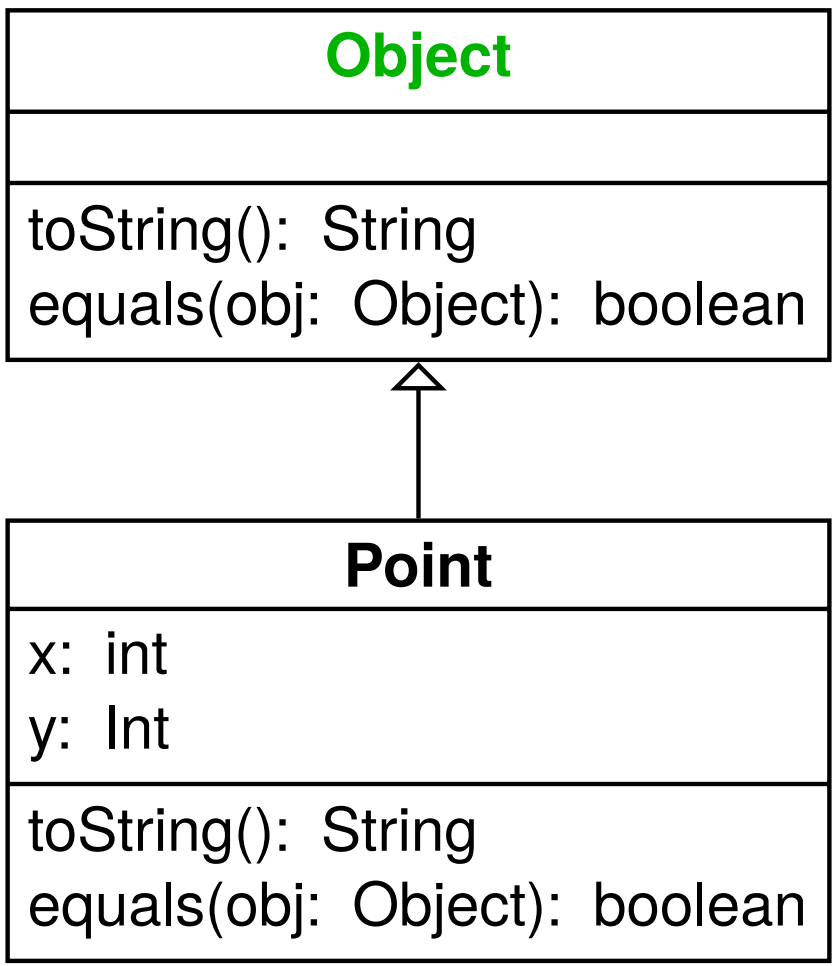


Utökad klass med extra attribut

```
1  package week07.superconstructor;
2
3  public class TalkingRobot2 extends Robot {
4      private String phrase;
5
6      public TalkingRobot2(String name, String phrase) {
7          super(name);  // call superclass constructor
8          this.phrase = phrase;
9      }
10
11     public void talk() {
12         System.out.println(name + " says: " + phrase);
13     }
14 }
```

```
1  package week07.superconstructor;
2
3  public class TestSuperConstructor2 {
4      public static void main(String[] args) {
5          TalkingRobot2 c3po = new TalkingRobot2("C-3PO", "I love humans!");
6          c3po.talk();
7          c3po.work();
8      }
9  }
```

Implementera innehållslikhet med `equals`



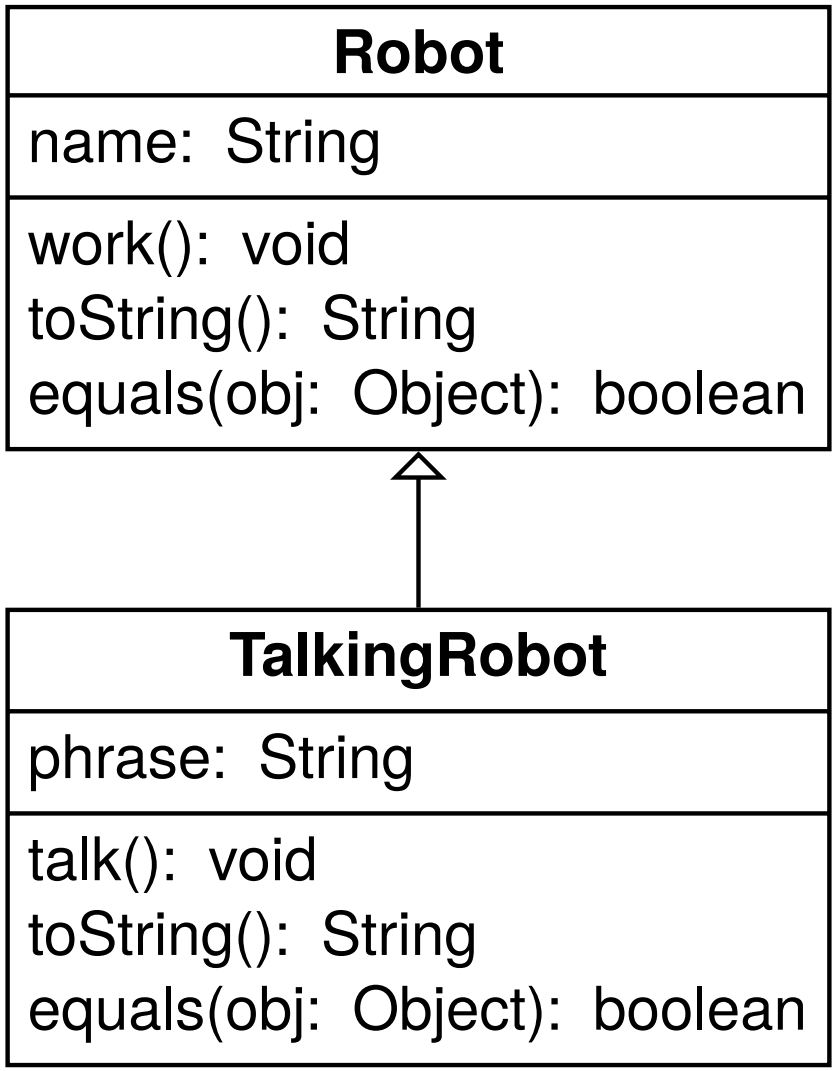
Implementera innehållslikhet för `Point` med `equals`

```
1  package week07.equals;
2
3  public class Point {
4      private int x;
5      private int y;
6
7      public Point(int x, int y) {
8          this.x = x;
9          this.y = y;
10     }
11
12     @Override  // tell the compiler to check that toString() actually exists in superclass
13     public String toString() {
14         return "Point [x=" + x + ", y=" + y + "]";
15     }
16
17     @Override
18     public boolean equals(Object obj) {
19         if (obj instanceof Point) {     // false if obj is null
20             Point other = (Point) obj;  // safe to cast obj to Point
21             return x == other.x && y == other.y;
22         } else {
23             return false;
24         }
25     }
26 }
```

Test av innehållslikhet för Point

```
1  package week07.equals;
2
3  public class TestEqualsPoint {
4      public static void equalityTest(Point a, Point b){
5          System.out.println("*** Testing equality of " + a + " and " + b);
6          if (a == b) {
7              System.out.println("    Reference equality: same");
8          } else {
9              System.out.println("    Reference inequality: different");
10         }
11         if (a.equals(b)){
12             System.out.println("    Structural equality: same contents");
13         } else {
14             System.out.println("    Structural inequality: different contents");
15         }
16     }
17
18     public static void main(String[] args) {
19         Point p1 = new Point(100, 100);
20         Point p2 = new Point(100, 100);
21         Point p3 = new Point(200, 100);
22         Point p4 = null;
23         equalityTest(p1,p2);
24         equalityTest(p1,p3);
25         equalityTest(p1,p4);
26         // equalityTest(p4,p4);  // run time error: NullPointerException
27     }
28 }
```

Implementera `equals` och `toString` vid arv



Implementera `toString` och `equals` i superklass

```
1  package week07.equals;
2
3  public class Robot {
4      protected String name;
5
6      public Robot(String name) {
7          this.name = name;
8      }
9
10     public void work() {
11         System.out.println(name + " is working.");
12     }
13
14     @Override
15     public String toString() {
16         return "Robot [name=" + name + "]";
17     }
18
19     @Override
20     public boolean equals(Object obj) {
21         if (obj instanceof Robot) {
22             Robot other = (Robot) obj;
23             return name.equals(other.name);
24         } else {
25             return false;
26         }
27     }
28 }
```

Implementera `toString` och `equals` i subklass

```
1  package week07.equals;
2
3  public class TalkingRobot extends Robot {
4      private String phrase;
5
6      public TalkingRobot(String name, String phrase) {
7          super(name);                // call constructor in superclass
8          this.phrase = phrase;       // extended initialization in this subklass
9      }
10
11     public void talk() {
12         System.out.println(phrase);
13     }
14
15     @Override
16     public String toString() { // call the toString() method in superclass
17         return "Talking" + super.toString() + " [phrase=" + phrase + "]";
18     }                          // what happens if you remove super. ???
19
20     @Override
21     public boolean equals(Object obj) {
22         if (obj instanceof TalkingRobot) {
23             TalkingRobot other = (TalkingRobot) obj;
24             return name.equals(other.name) && phrase.equals(other.phrase);
25         } else {
26             return false;
27         }
28     }
29 }
```

Test av innehållslikhet för Robot och TalkingRobot

```
1  package week07.equals;
2
3  public class TestEqualsRobot {
4      public static void equalityTest(Object a, Object b){
5          System.out.println("*** Testing equality of " + a + " and " + b);
6          if (a == b) {
7              System.out.println("    Reference equality: same");
8          } else {
9              System.out.println("    Reference inequality: different");
10         }
11         if (a.equals(b)){
12             System.out.println("    Structural equality: same contents");
13         } else {
14             System.out.println("    Structural inequality: different contents");
15         }
16     }
17
18     public static void main(String[] args) {
19         Robot r1 = new Robot("Wall*E");
20         Robot r2 = new TalkingRobot("Marvin", "I am very depressed.");
21         Robot r3 = new TalkingRobot("Marvin", "I am very depressed.");
22         Point p1 = new Point(100, 100);
23         equalityTest(r1,r1);
24         equalityTest(r1,r2);
25         equalityTest(r2,r3);
26         equalityTest(r1,p1);
27     }
28 }
29 //      https://www.youtube.com/watch?v=YKd6ZxCEJdI
```