```java
public interface Skrivbar {
  void skriv();
}


public class Punkt implements Skrivbar {
  public double x;
  public double y;

  public Punkt(double xx, double yy) {
    x = xx; y = yy;
  }

  public Punkt() {}          // defaultkonstruktor

  @Override
  public String toString() {
    return "(" + x + ", " + y + ")";
  }

  public void skriv() {
    System.out.println(toString());
  }
}
```

```
public interface Ritbar {
   void rita();
}


public class Punkt implements Skrivbar, Ritbar {
  som tidigare
  public void skriv() {
    System.out.println(toString());
  }
  public void rita() {
    här ligger satser som ritar en punkt
  }
}
```

```
public abstract class GrafiskFigur {
  deklarationer av variabler om metoder
}

public class Punkt extends GrafiskFigur
                   implements Skrivbar, Ritbar {
  som tidigare
}


public abstract class Djur implements Ritbar {
  deklarationer av variabler om metoder
}
```

```
Skrivbar sk = new Skrivbar();  // FEL!!


// Men följande är OK
Punkt p = new Punkt(1.0, 2.5);
Tiger ti = new Tiger();
Skrivbar sk;
Ritbar ri1, ri2;
sk = p;
ri1 = p;
ri2 = ti;
ri2.rita();  // dynamisk bindning
```

```
public interface Presenterbar extends Ritbar, Skrivbar {
  void visa();
}


public interface Lagringsbar {
  int blockStorlek = 1024;    // automatiskt final och static
  void lagra();
}
```

```
Ritbar[] r = new Ritbar[100];
r[0] = new Tiger();
r[1] = new Punkt();
r[2] = new Mus();


for (int i=0; i<r.length; i++)
  if (r[i] != null)
    r[i].rita();
```

Nyhet i Java 8: default-metoder (defender methods, virtual extension methods)

```
public interface Skrivbar {
  default void skriv(){
    System.out.println(toString());
  }
}
```

Nu OK:

```
public class Punkt implements Skrivbar {
  public double x;
  public double y;

  public Punkt(double xx, double yy) {
    x = xx; y = yy;
  }

  public Punkt() {}          // defaultkonstruktor

  @Override
  public String toString() {
    return "(" + x + ", " + y + ")";
  }
}
```

Default-metoder

- krånglar till reglerna för interface
- gör det möjligt att utöka ett interface utan att befintliga klasser som implementerar det behöver påverkas.
- gäller bara om ingen implementering skett på ”normalt” sätt (egen implementering eller arv från superklass)
- upphävs om de omdefineras i ett subgränssnitt (även om de där inte är default-metoder)
- ger oklarheter när en viss metod finns i mer än ett gränssnitt vilka implementeras av en klass eller utökas i ett subgränssnitt. (Kan lösas med hjälp av `super`.)

Nyhet 2 i Java 8: klass-metoder (static)

```
public interface Fordon {
   default void skriv() {
     System.out.println("Ett fordon :" + toString());
   }
   int avläsHastighet();
   static int jämförHastighet(Fordon f1, Fordon f2) {
     return f1.avläsHastighet()-f2.avläsHastighet();
   }
}
```

```
if (Fordon.jämförHastighet(c,m) > 0)
  System.out.println("Cykeln är snabbast");
```

Jämförelser

```
public class Klockslag implements Comparable<Klockslag> {
  int tim;
  int min;

  public int compareTo(Klockslag k) {
    if  (tim < k.tim || (tim == k.tim && min < k.min))
      return -1;
    else if (tim > k.tim || (tim == k.tim && min > k.min))
      return 1;
    else
      return 0;
  }

  @Override
  public String toString() {
    return String.format("%02d:%02d", tim, min);
  }
}
```

```
int i = k1.compareTo(k2);
if (i < 0)
  System.out.println(k1 + " kommer före " + k2);
else if (i == 0)
  System.out.println("Samma klockslag");
else
  System.out.println(k1 + " kommer efter " + k2);
```

```
public class Cirkel {
  // instansvariabler
  double x, y;     // mittpunktens koordinater
  double radie;

  // instansmetoder
  public void sättRadie(double r) {  // ändrar radien
    if (r >= 0)
      radie = r;
    else
      throw new IllegalArgumentException("Negativ radie");
  }
  public double area() {             // beräknar arean
    return Math.PI * radie * radie;
  }
  public double omkr() {             // beräknar omkretsen
    return 2 * Math.PI * radie;
  }
}
```

```
import java.util.*;     // innehåller bl.a. Comparator
public class JfrCirkel implements Comparator<Cirkel> {
  public int compare(Cirkel a, Cirkel b) {
    if (a.radie < b.radie)
      return -1;
    else if (a.radie > b.radie)
      return 1;
    else
      return 0;
  }
}


JfrCirkel jfr = new JfrCirkel();   // skapa en jämförare
if (jfr.compare(c1,c2) < 0)
  JOptionPane.showMessageDialog(null, "Den första cirkeln är minst");
```

```
import java.text.*;   // Innehåller Collator
public class Namn implements Comparable<Namn> {
  String förnamn;
  String efternamn;
  static Collator col = Collator.getInstance();
  public int compareTo(Namn annatNamn) {
    col.setStrength(Collator.PRIMARY);
    int i = col.compare(efternamn, annatNamn.efternamn);
    if (i != 0)
      return i;
    else
      return col.compare(förnamn, annatNamn.förnamn);
  }
  @Override
  public String toString() {
    return förnamn + " " + efternamn;
  }
  ...
}
```

Några metoder i klassen java.util.Arrays

`toString(a)`	ger en textversion av fältet `a`. Om komponenterna är objekt anropas `toString` för varje komponent.
`equals(a,b)`	ger **`true`** om fälten `a` och `b` är lika, dvs. har lika många komponenter och motsvarande komponenter är lika (`a[i].equals(b[i])` om komponenterna är objekt)
`sort(a)`	sorterar komponenterna i fältet `a`
`sort(a, c)`	som ovan, men använder en jämförare `c`
`sort(a,i1,i2)`	sorterar komponenterna nr `i1` till `i2-1` i fältet `a`
`sort(a,i1,i2,c)`	som ovan, men använder en jämförare `c`
`binarySearch(a,k)`	söker efter komponenten `k` i fältet `a`, ger `k`:s index i fältet om `k` finns, -1 annars
`binarySearch(a,k,c)`	som ovan, men använder en jämförare `c`
`asList(a)`	ger möjlighet att hantera fältet `a` som en lista

Några metoder i klassen java.util.Collections

`sort(l)`	sorterar elementen i listan `l`.
`sort(l, c)`	som ovan, men använder en jämförare `c`.
`binarySearch(l,e)`	söker efter elementet `e` i listan `l`, ger `k`:s index i listan om `k` finns, -1 annars.
`binarySearch(l,e,c)`	som ovan, men använder en jämförare `c`.
`max(l)`	ger det största elementen i listan `l`.
`max(l,c)`	som ovan, men använder en jämförare `c`.
`min(l)`	ger det minsta elementen i listan `l`.
`min(l,c)`	som ovan, men använder en jämförare `c`.