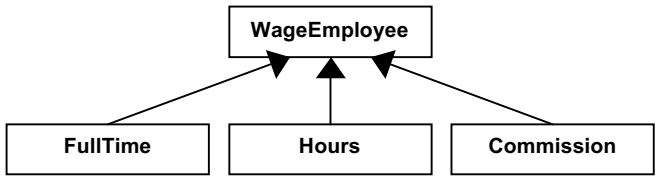
Laboration 15 – Polymorfism, Interface

Avsikten med laboration 15 är att du ska använda polymorfism och implementera interface. Speciellt ska du implementera interfacen *Comparable*, *Comparator* och *Icon*. Konsultera facit när du är färdig med en uppgift. Placera java-filerna i paketet ***laboration15***. Placera bildfilerna (jpg) i katalogen *images* i projektkatalogen.

Grundläggande uppgifter

Uppgift 1

I ett företag avlönas de anställda enligt tre olika modeller – fast månadslön, timanställning och provision på försäljning. Företaget använder en klass för varje lönemodell och dessa klasser ärver den abstrakta klassen ***WageEmployee***:



Testkör programmet ***Employed.java*** (finns på kurssdian) med hjälp av debuggern. Sätt en breakpoint på rad 6 och starta sedan programmet med ***Debug As – Java Application*** (debuggern).

```
1  package laboration23;
2
3  public class Employees {
4      public void program() {
5          Commission cWage;
6          WageEmployee[] employees = new WageEmployee[ 3 ];
7          employees[ 0 ] = new FullTime( 19938278, 21500 );
8          employees[ 1 ] = new Commission( 19278865, 0.10 );
9          cWage = (Commission)employees[ 1 ];
10         cWage.setSales( 208000 );
11         employees[ 2 ] = new Hours( 17233534, 95.0 );
12         (( Hours )employees[ 2 ]).setHours( 128 );
13         for( int i = 0; i < employees.length; i++ ) {
14             System.out.println( employees[ i ] );
15         }
16     }
17     public static void main( String[] args ) {
18         Employees prog = new Employees();
19         prog.program();
20     }
21 }
```

Stega dig framåt i programmet med hjälp av ***Step Over*** (F6). Besvara följande frågor:

1. Stega framåt så att rad 6 exekverats. Vilken typ av objekt refererar variabeln *employees* till när rad 6 exekverats? Hur många element kan lagras i arrayen? Vilket värde har elementen i arrayen?
2. Stega framåt så att rad 7 exekverats. Vad händer på rad 7?
3. Stega framåt så att rad 8 exekverats. Vad händer på rad 8?
4. Stega framåt så att rad 9 och rad 10 exekverats. Vad händer på dessa rader?
5. Stega framåt så att rad 11 exekverats. Vad händer på rad 11?
6. Stega framåt så att rad 12 exekverats. Vad händer på rad 12?
7. Stega framåt i for-loopen på raderna 13-15. Vad utförs varje upprepning (dvs rad 14)

Uppgift 2

Nu ska du implementera interfacet ***Comparable*** i klassen ***WageEmployee***.

```
public class WageEmployee implements Comparable {

    // all kod sedan tidigare

    public int compareTo( Object obj ) {
        // se beskrivning nedan
    }
}
```

Metoden ***compareTo*** ska jämföra lönen i två WageEmployee-objekt. Detta sker genom att lönen i det aktuella objektet (objektet som anropar compareTo-metoden) jämförs med lönen i objektet som är parameter. Metoden ska göra följande:

1. Deklarera variabeln *emloyed* av typen *WageEmployee*
2. Tilldela variabeln *employed* objektet som *obj* refererar till (ett objekt som ärver WageEmloyee) dvs:
   ```
   employed = ( WageEmployee )obj;
   ```
3. Jämföra det aktuella objektets lön med lönen i objektet *employed*. Vardera objektets lön får du genom anrop till metoden wage(), dvs:
   ```
   double wage1 = this.wage();     // Lönen i aktuellt objekt
   double wage2 = employed.wage(); // Lönen i parameter-objektet
   ```
4. Om wage1 < wage2 så returnera värdet -1
   Om wage1 > wage2 så returnera värdet 1
   Om lönerna är lika stora returnera värdet 0

Exempel (javax.swing.* och java.util.* måste importeras):

```
WageEmployee[] employees = { new FullTime(19938278, 21500),
        new Commission(19278865, 0.10), new Hours(17233534, 95.0) };
String res = "FÖRE SORTERING\n";
((Commission) employees[1]).setSales(208000);
((Hours) employees[2]).setHours(128);
for (int i = 0; i < employees.length; i++) {
    res += employees[i].toString() + "\n";
}
Arrays.sort(employees);
res += "-----------------------------------\n";
res += "EFTER SORTERING\n";
for (int i = 0; i < employees.length; i++) {
    res += employees[i].toString() + "\n";
}
JOptionPane.showMessageDialog(null, res);
```

Du ser **körresultatet** till höger.

När du löser uppgifterna nedan (där du ska implementera interfacet ***Icon***) så kan du utgå från en mall liknande denna:

```
package laboration15;
import java.awt.*;
// import java.awt.geom.*;        // Behövs vid anrop till draw/fill
// import javax.swing.*;          // Behövs om du ska använda bilder
// import java.awt.image.*;       // Behövs om du ska använda bilder

public class Exercise3 implements Icon {
    // Instansvariabler som används i programmet

    // Metoder som används i programmet

    public void paintIcon(Component c, Graphics g, int x, int y) {
        // Graphics2D g2 = (Graphics2D)g; // Uppgift 4, ...

        // Ritkommandon, metodanrop mm här
    }

    public int getIconWidth() {
        return AAA; // Ersätt AAA med bildens bredd
    }

    public int getIconHeight() {
        return AAA; // Ersätt AAA med bildens höjd
    }

    public static void main(String[] args)  {
        IconWindow.showIcon( new Exercise3() );
    }
}
```

Uppgift 3

Testa ritkommandona som finns i Graphics-objektet. Du kan rita vad du vill, jag gjorde en ö i havet:

Du ska alltså ersätta AAA i metoderna *getIconWidth* och *getIconHeight* i mallen ovan (bildens bredd och höjd). Sedan lägger du till ritkommandon i *paintIcon*-metoden.

Uppgift 4

Det är dags att tillverka lite "modern konst". Den moderna konsten består av ett antal slumpmässigt dragna linjer. Följande är slump:

- Startpunkt *(x1, y1)* och slutpunkt *(x2,y2)* för linjen. Båda punkterna ska slumpas så de är inuti fönstret.
- Färgen är slumpmässig (*new Color(red, greeen, blue)*) där *red*, *green* och *blue* har slumvärden i intervallet 0-255.
- Linjens bredd (*width*) är slumpmässig och i intervallet 4 – 20.

Sedan ritas linjen med ett Graphics2D-objekt:

```
g2.setPaint( new Color( red, green, blue ) ); // ritfärg
g2.setStroke( new BasicStroke( width ) );     // bredd
g2.draw( new Line2D.Double(x1, y1, x2, y2) ); // från - till
```

Ca 20 slumplinjer kan vara lagom. Låt antalet linjer vara argument då klassen skapas.

Uppgift 5

I denna uppgift ska en klass implementera interfacet ***WageFilter***:

```
package laboration15;

public interface WageFilter {
    public boolean accept( WageEmployee employed );
}
```

Klassen ***Max20000*** implementerar interfacet ***WageFilter*** på så sätt att om ett *WageEmployee*-objekt har en lön som är högst 20000 så returneras *true*, annars *false*.

```
package laboration15;

public class Max20000 implements WageFilter {
    public boolean accept( WageEmployee employed ) {
        double wage = employed.wage();
        return ( wage <= 20000 );
    }
}
```

Programmet ***TestWageFilter*** demonstrerar hur klassen *Max20000* kan användas för att välja ut de *WageEmployee*-objekt vars lön är högst 20000 kr.

```
package laboration15;

public class TestWageFilter {
    public static void main( String[] args ) {
        WageFilter filter = new Max20000();
        WageEmployee[] employees = { new FullTime( 19938278, 21500 ),
               new Commission( 19278865, 0.10 ),
               new Hours( 17233534, 95.0 ),
               new FullTime( 27348263, 18000 )};
        ((Commission)employees[ 1 ]).setSales( 208000 );
        (( Hours )employees[ 2 ]).setHours( 128 );
        for( int i = 0; i < employees.length; i++ ) {
            if( filter.accept( employees[ i ] ) ) {
                System.out.println( employees[ i ] );
            }
        }
    }

}
```

När du kör programmet får du utskrifterna:

```
Id: 17233534, lön denna månad: 12160.0 kr
Id: 27348263, lön denna månad: 18000.0 kr
```

Uppgift 5a

Skriv klassen ***Between17000And21000*** vilken ska implementera interfacet *WageFilter*. Metoden *accept* ska returnera *true* om lönen är i intervallet 17000 – 21000 och annars *false*.

Om du testar din klass med *TestWageFilter* så ska du ersätta den första raden med:

```
WageFilter filter = new Between17000And21000();
```

Sedan ska du erhålla utskrifterna:

```
Id: 19278865, lön denna månad: 20800.0 kr
Id: 27348263, lön denna månad: 18000.0 kr
```

Uppgift 5b

Skriv klassen ***Interval*** vilken ska implementera interfacet *WageFilter*. Metoden *accept* ska returnera *true* om lönen är inom ett intervall vilket anges vid konstruktion, t.ex.

```
WageFilter filter = new Interval( 19000, 21000 );
```

Om du kör *TestWageFilter* med ovanstående rad så ska du få utskriftera:

```
Id: 19278865, lön denna månad: 20800.0 kr
```

Uppgift 6

Klassen ***DiscardIcon*** ritar ett rött kryss över ett *Icon*-objekt (se DiscardIcon.java).

Konstruktorn i klassen *DiscardIcon* tar ett *Icon*-objekt som argument. Referensen till *Icon*-objektet sparas i instansvariabeln *icon*.

```
public class DiscardIcon implements Icon {
    private Icon icon;

    public DiscardIcon(Icon iconObject) {
        this.icon = iconObject;
    }
```

Instansvariabeln *icon* används i metoderna *paintIcon*, *getIconWidth* och *getIconHeight*

I ***paintIcon*** ritas först Icon-objektet och därefter ett rött kryss. Krysset ritas med hjälp av en for-loop. Två linjer ritas varje upprepning.

```
public void paintIcon(Component c, Graphics g, int x, int y) {
    icon.paintIcon(c,g,x,y); // Icon-objektet ritas
    g.setColor(Color.RED);   // Ändra ritfärg till röd
    for(int i=-2; i<=2; i++) {
        g.drawLine(x, y+i, x+icon.getIconWidth(), y+icon.getIconHeight()+i);
        g.drawLine(x, y+icon.getIconHeight()+i, x+icon.getIconWidth(), y+i);
    }
}
```

Metoden ***getIconWidth/getIconHeight*** returnerar Icon-objektets bredd/höjd (ändras ej)

```
public int getIconWidth() {
    return icon.getIconWidth();
}
```

```
public int getIconHeight() {
    return icon.getIconHeight();
}
```

Testkör ikonen med raderna

```
ImageIcon icon = new ImageIcon("images/Gubbe.jpg");
IconWindow.showIcon(new DiscardIcon(icon)); // Överkryssad gubbe
```

Skriv klassen ***BorderIcon*** vilken ska rita en ram runt ett Icon-objekt. Från början kan du låta ramen vara röd och 8 pixlar bred. Sedan kan du låta dessa värden vara argument vid konstruktion.

Att tänka på när du skriver BorderIcon är att:

användaren ska ange ikonen som ska ramas in i konstruktorn. I den slutgilitga versionen ska även färg på ram och ramens bredd vara parametrar i konstruktorn.

```
public BorderIcon(Icon icon)
```

Bredden på den nya Ikonen blir bredare än den ursprungliga. Om ramen är 8 pixlar bred så måste getIconWidth/getIconHeight öka bredden/höjden med 16.

När du ska rita Icon-objektet så måste det ritas 8 pixlar ner och 8 pixlar åt höger så att ramen får plats.

När du ritar ramen kan du använda en for-loop som upprepas 8 gånger.

Första iterationen ritar du en rektangel längst ut:

```
g.drawRect(x,y,newWidth, nyHeight);
```

Andra iterationen ritar du en rektangel precis inuti den föregående:

```
g.drawRect(x+1,y+1,nybredd-2,nyHöjd-2);
```

osv.

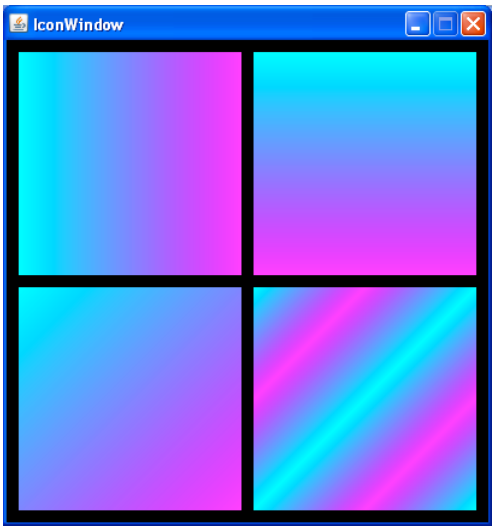
Uppgift 7

Skriv metoden *collect(WageEmployee[] employees, WageFilter filter)* vilken returnerar en ny array med de *WageEmployee*-objekt där *WageFilter*-objektet returnerar *true*.

Uppgift 8

I uppgift 8 ska du använda ***GradientPaint*** för att fylla några rektanglar. I samtliga fall är det färgerna *Color.CYAN* och *Color.MAGENTA* som används.

När *GradientPaint*-objekt skapas anger man 6 eller 7 argument: *x1, y1, color1, x2, y2, color2, repeat*



- Punkterna *(x1, y1)* och *(x2, y2)* anger ett par saker:
  - Hur lång övergången från den ena färgen till den andra ska vara. I den ena punkten börjar övergången och i den andra slutar den.
  - Hur riktningen på mönstret ska vara. Är punkterna på samma höjd så blir mönstret vertikalt. Är punkterna under varandra blir mönstret horisontellt osv.
- Övergången startar med *color1* och slutar med *color2*.
- Det sista argumentet, *repeat*, anger om övergången ska upprepas så länge det är mer yta att fylla. Om argumentet anges till *true* så kommer upprepning ske. Om argumentet anges till *false* eller utelämnas så sker endast en övergång.

Fyll rektanglar så de liknar rektanglarna i figuren.

Innan du anropar fill-metoden ska du alltså skapa ett *GradientPaint*-objekt och anropa metoden *setPaint*:

```
Graphics2D g2 = (Graphics2D)g;
Paint paint = new GradientPaint(x+x1, y+y1, Color.CYAN, x+x2, y+y2, Color.MAGENTA);
g2.setPaint( paint );
g2.fill( new Rectangle2D.Double( x+xPos, y+yPos, width, height ) );
```

Extrauppgifter

Uppgift 9

I uppgift 9 ska du skriva klassen **ResizeIcon** vilken ska visa en ikon men med ny storlek. Vid konstruktion ska tre argument ges: *icon* som ska användas, *ny bredd* och *ny höjd*.

Ett sätt att lösa uppgiften är att låta klassen ha en instansvariabel av typen *BufferedImage*. Det är också lämpligt att ha den nya ikonens dimensioner som instansvariabler (width resp height)

I konstruktorn skapar du *BufferedImage*-objektet med samma dimensioner som den ursprungliga ikonen. Sedan ritar du ikonen i *BufferedImage*-objektet. Detta gör du genom att anropa icon.paintIcon-metoden. Som första argument kan du använda null. Det andra argument ska vara grafikverktyget till *BufferedImage*-objektet. De två sista argumenten ska båda vara 0.

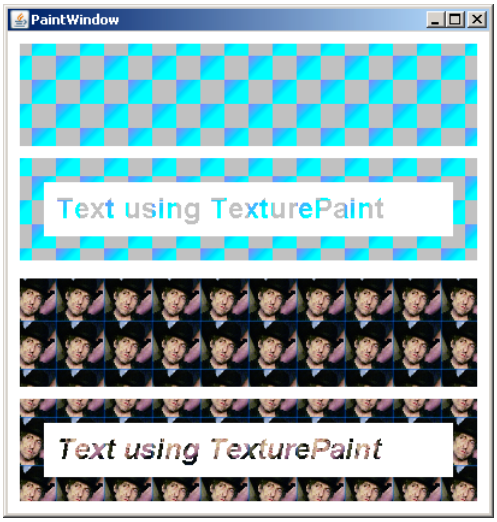
Metoden *getIconWidth* och *getIconHeight* ska returnera den nya ikonens bredd resp höjd.

Metoden *paintIcon* ska rita *BufferedImage*-objektet. Det gör du på samma sätt som när du ritar *ImageIcon*-objekt. Men använd *drawImage*-metoden med 6 argument: *image, x, y, width, height, null*

När du är färdig med klassen kan du testa den med följande kod. Varför har inte ikonerna samma storlek och utseende?

```
ImageIcon icon = new ImageIcon("images/Gubbe.jpg");
Icon resize = new ResizeIcon(icon,200,50);
Icon border = new BorderIcon(icon,Color.BLUE,5);
IconWindow.showIcon(new ResizeIcon(border,200,50));
IconWindow.showIcon(new BorderIcon(resize,Color.BLUE,5));
```

Uppgift 10



I uppgift 10 ska du använda ***TexturePaint*** för att fylla några rektanglar och skriva lite text. *TexturePaint* utgår från en bild (av typen *BufferedImage*) vilken upprepas tills ytan är fylld. Stegen för att skapa en *TexturePaint* är

1. Skapa en *BufferedImage* av lämplig storlek
2. Hämta grafikverktyg till *BufferedImage*-objektet
3. Rita bilden som ska användas
4. Skapa ett *TexturePaint*-objekt med hjälp av bilden du skapat

Ovanstående steg motsvars ungefär av nedanstående rader med kod:

```
1.  BufferedImage bi = new BufferedImage( 40, 40,
    BufferedImage.TYPE_INT_RGB );
2.  Graphics2D bi2 = (Graphics2D)bi.getGraphics();

3.  bi2.setPaint( Color.RED );
    bi2.fill( new Rectangle2D.Double( x, y, 20, 20 ) );
    // fler ritkommandon
4.  paint = new TexturePaint( bi, new Rectangle( 40, 40 ) );
```

Nu kan du anropa *setPaint* och fylla ytor. Om du vill använda en ***bild*** i din *TexturePaint* så ska du i vanlig ordning hämta bilden med hjälp av ett *ImageIcon*-objekt. Sedan ritar du bilden i *BufferedImage*-objektet med *drawImage*-metoden (bi2.drawImage(…);)

Om du skriver ***text*** med *drawString*-metoden kommer texten visas med hjälp av *Paint*-objektet (Color / GradientPaint / TexturePaint).
I figuren användes fonten "SansSerif", fet, 24 punkter resp "SansSerif", fet+kursiv, 24 punkter

Lösningar

Uppgift 1

1. *employees* refererar till en array som kan lagra 3 st *WageEmloyee*-objekt, dvs objekt som är subklass till *WageEmployee*. Från början har elementen värdet *null* eftersom det inte lagrats några objekt i arrayen.
2. Ett objekt av typen *FullTime* skapas. Det första elementet i arrayen *employees* (dvs position 0) tilldelas referens till objektet.
3. Ett objekt av typen *Commission* skapas. Det andra elementet i arrayen *employees* (dvs position 1) tilldelas referens till objektet.
4. Referensvariabeln *cWage* tilldelas referensen till det objekt som lagras i position 1 i arrayen. Men för att detta ska vara möjligt måste referensen typkonverteras till ”*referens till Commission-objekt*”. Detta går bra eftersom objektet som lagras i position 1 är av typen *Commission*. Sedan ändras egenskapen *sales* i objektet till 208000.
5. Ett objekt av typen *Hours* skapas. Det tredje elementet i arrayen *employees* (dvs position 2) tilldelas referens till objektet.
6. Referensen i *emlpoyees[2]* typkonverteras till ”*referens till Hours-objekt*”. Efter typkonverteringen används referensen för att anropa metoden *setHours*. Egenskapen *hours* ändras då till 128.
7. Varje gång rad 14 exekveras så anropas *toString*-metoden (i *WageEmployee*) för ett av objekten i *employees*-arrayen. Strängen som returneras vid anropet skrivs ut.

Uppgift 2

```
package laboration15;

public abstract class WageEmployee implements Comparable {
    private long id;

    public WageEmployee( long id ) {
        this.id = id;
    }

    public long getId() {
        return this.id;
    }

    public String toString() {
        return "Id: " + this.id + ", lön denna månad: " + wage() + " kr";
    }

    public abstract double wage();

    public int compareTo( Object obj ) {
        WageEmployee employed;
        employed = ( WageEmployee )obj;
        double wage1 = this.wage();
        double wage2 = employed.wage();
        if( wage1 < wage2 ) {
            return -1;
        } else if( wage1 > wage2 ) {
            return 1;
        } else {
            return 0;
        }
```

```
    }
}
```

Uppgift 3

```
package laboration15;
import java.awt.*;
import javax.swing.Icon;

public class Exercise3 implements Icon {
    private Color brown = new Color(123, 62, 41);
    private Color darkGreen = new Color(8, 78, 8);

    private void palm(Graphics g, int x, int y) {
        int[] palmx = {x+2, x+10, x+6, x+17, x+7, x + 12, x+2, x-11, x-3, x-6};
        int[] palmy = {y+2, y+6, y-2, y-7, y-4, y-12, y-5, y-9, y-2, y+8};
        g.setColor(brown);
        g.fillRect(x, y, 5, 13);
        g.setColor(darkGreen);
        g.fillPolygon(palmx, palmy, 9);
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        g.setColor(Color.BLUE);
        g.fillRect(x,y+220,getIconWidth(),getIconHeight()-220);

        g.setColor(Color.CYAN);
        g.fillRect(x, y, 600, 220);
        g.setColor(Color.GREEN);
        g.fillOval(x + 300, y + 230, 200, 30);
        g.fillOval(x + 350, y + 210, 130, 50);
        g.setColor(Color.ORANGE);
        g.fillOval(x + 100, y + 100, 70, 70);
        for (int i = 0; i < 40; i++) {
            palm(g, x+350+(int)(Math.random()*40-i/3*2), y+220+i/2);
        }
        for (int i = 0; i < 40; i++) {
            palm(g, x+470-(int)(Math.random()*40+i), y+210+i/2);
        }
    }

    public int getIconWidth() {
        return 600;
    }

    public int getIconHeight() {
        return 400;
    }

    public static void main(String[] args) {
        IconWindow.showIcon(new Exercise3());
    }
}
```

Uppgift 4

```
import java.awt.geom.*;
import java.awt.*;
import javax.swing.Icon;

public class Exercise4 implements Icon {
    private int count;

    public Exercise4(int count) {
        this.count = count;
    }

    public int getSlump(int min, int max) {
        return (int) (Math.random() * (max - min + 1)) + min;
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        int x1, y1, x2, y2, width, i = 0;
        Stroke stroke;
        Paint paint;
        Graphics2D g2 = (Graphics2D) g;
        g.setColor(Color.BLACK);
        g.fillRect(x, y, getIconWidth(), getIconHeight());
        while (i < this.count) {
            x1 = getSlump(10, 390);
            y1 = getSlump(10, 390);
            x2 = getSlump(10, 390);
            y2 = getSlump(10, 390);
            width = getSlump(4, 20);
            stroke = new BasicStroke(width);
            paint = new Color(getSlump(0,255), getSlump(0,255), getSlump(0,255));
            g2.setStroke(stroke);
            g2.setPaint(paint);
            g2.draw(new Line2D.Double(x + x1, y + y1, x + x2, y + y2));
            i++;
        }
    }

    public int getIconWidth() {
        return 400;
    }

    public int getIconHeight() {
        return 400;
    }

    public static void main(String[] args) {
        IconWindow.showIcon(new Exercise4(20));
    }
}
```

Uppgift 5a

```
public class Between17000And21000 implements WageFilter {
    public boolean accept( WageEmployee employed ) {
        double wage = employed.wage();
        return ( wage >= 17000 ) && ( wage <= 21000 );
    }
}
```

Uppgift 5b

```
public class Interval implements WageFilter {
    private double min;
    private double max;

    public Interval( double min, double max ) {
        this.min = min;
        this.max = max;
    }

    public boolean accept( WageEmployee employed ) {
        double wage = employed.wage();
        return ( wage >= min ) && ( wage <= max );
    }
}
```

Uppgift 6

Uppgift 6, verison 1

```
package laboration15;
import java.awt.*;
import javax.swing.*;

public class BorderIcon implements Icon {
    private Icon icon;

    public BorderIcon(Icon icon) {
        this.icon = icon;
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        icon.paintIcon(c, g, x+8, y+8);
        g.setColor(Color.RED);
        for(int i=0; i<8; i++) {
            g.drawRect(x+i, y+i, getIconWidth()-2*i, getIconHeight()-2*i);
        }
    }

    public int getIconWidth() {
        return icon.getIconWidth() + 2*8;
    }

    public int getIconHeight() {
        return icon.getIconHeight() + 2*8;
    }

    public static void main(String[] args) {
        ImageIcon icon = new ImageIcon("images/Gubbe.jpg");
        IconWindow.showIcon(new BorderIcon(icon));
    }
}
```

Uppgift 6, version 2

```
public class BorderIcon implements Icon {
    private Icon icon;
    private Color color;
    private int width;

    public BorderIcon(Icon icon, Color color, int width) {
        this.icon = icon;
        this.color = color;
        this.width = width;
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        icon.paintIcon(c, g, x+width, y+width);
        g.setColor(color);
        for(int i=0; i<width; i++) {
            g.drawRect(x+i, y+i, getIconWidth()-2*i, getIconHeight()-2*i);
        }
    }

    public int getIconWidth() {
        return icon.getIconWidth() + 2*width;
    }

    public int getIconHeight() {
        return icon.getIconHeight() + 2*width;
    }

    public static void main(String[] args) {
        ImageIcon icon = new ImageIcon("images/Gubbe.jpg");
        IconWindow.showIcon( new BorderIcon(icon,Color.RED,8));
    }
}
```

Uppgift 7

```
public WageEmployee[] collect(WageEmployee[] employees, WageFilter filter ) {
    int counter = 0, index = 0;
    WageEmployee[] res;
    for( int i = 0; i < employees.length; i++ ) {
        if( filter.accept( employees[ i ] ) ) {
            counter++;
        }
    }
    res = new WageEmployee[ counter ];
    for( int i = 0; i < employees.length; i++ ) {
        if( filter.accept( employees[ i ] ) ) {
            res[ index ] = employees[ i ];
            index++;
        }
    }
    return res;
}
```

Uppgift 8

```
import java.awt.*;
import java.awt.geom.*;
import javax.swing.Icon;

public class Exercise8 implements Icon {
    private void horisontal(Graphics g, int x, int y, int width, int height) {
        Graphics2D g2 = (Graphics2D) g;
        Paint paint = new GradientPaint(x,y,Color.CYAN,x+width,y,Color.MAGENTA);
        g2.setPaint(paint);
        g2.fill(new Rectangle2D.Double(x, y, width, height));
    }

    private void vertical(Graphics g, int x, int y, int width, int height) {
        Graphics2D g2 = (Graphics2D) g;
        Paint paint = new GradientPaint(x,y,Color.CYAN,x,y+height,Color.MAGENTA);
        g2.setPaint(paint);
        g2.fill(new Rectangle2D.Double(x, y, width, height));
    }

    private void cornerToCorner1(Graphics g, int x, int y, int width, int height) {
        Graphics2D g2 = (Graphics2D) g;
        Paint paint = new GradientPaint(x,y,Color.CYAN,x+width,y+height, Color.MAGENTA);
        g2.setPaint(paint);
        g2.fill(new Rectangle2D.Double(x, y, width, height));
    }

    private void cornerToCorner4(Graphics g, int x, int y, int width, int height) {
        Graphics2D g2 = (Graphics2D) g;
        Paint paint=new GradientPaint(x,y,Color.CYAN,x+width/4,y+height/4,Color.MAGENTA,true);
        g2.setPaint(paint);
        g2.fill(new Rectangle2D.Double(x, y, width, height));
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        g.setColor(Color.BLACK);
        g.fillRect(x, y, getIconWidth(), getIconHeight());
        horisontal(g, x + 10, y + 10, 185, 185);
        vertical(g, x + 205, y + 10, 185, 185);
        cornerToCorner1(g, x + 10, y + 205, 185, 185);
        cornerToCorner4(g, x + 205, y + 205, 185, 185);
    }

    public int getIconWidth() {
        return 400;
    }

    public int getIconHeight() {
        return 400;
    }

    public static void main(String[] args) {
        IconWindow.showIcon(new Exercise8());
    }
}
```

Uppgift 9

```
import java.awt.*;
import java.awt.image.BufferedImage;
import javax.swing.*;

public class ResizeIcon implements Icon {
    private BufferedImage bi;
    private int width;
    private int height;

    public ResizeIcon(Icon icon, int width, int height) {
        this.width = width;
        this.height = height;
        bi = new BufferedImage(icon.getIconWidth(), icon.getIconHeight(),
BufferedImage.TYPE_INT_RGB);
        Graphics g = bi.getGraphics();
        icon.paintIcon(null, g, 0, 0);
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        g.drawImage(bi, x, y, width, height, null);
    }

    public int getIconWidth() {
        return width;
    }

    public int getIconHeight() {
        return height;
    }

    public static void main(String[] args) {
        ImageIcon icon = new ImageIcon("images/Gubbe.jpg");
        Icon resize = new ResizeIcon(icon,200,50);
        IconWindow.showIcon(resize);
    }
}
```

Uppgift 10

```
import java.awt.*;
import java.awt.geom.Rectangle2D;
import java.awt.image.BufferedImage;
import javax.swing.*;

public class Exercise10 implements Icon {

    private void texture(Graphics g, int x, int y) {
        // Skapa en 40x40 pixlar stor bild i minnet
        BufferedImage bi = new BufferedImage(40, 40, BufferedImage.TYPE_INT_RGB);
        // Grafikverktyg till bilden
        Graphics2D bi2 = (Graphics2D) bi.getGraphics();
        // Rita på bilden
        bi2.setPaint(Color.LIGHT_GRAY);
        bi2.fill(new Rectangle2D.Double(0, 0, 20, 20));
        bi2.fill(new Rectangle2D.Double(20, 20, 20, 20));
        bi2.setPaint(new GradientPaint(0, 0, Color.MAGENTA, 20, 20, Color.CYAN));
        bi2.fill(new Rectangle2D.Double(0, 20, 20, 20));
        bi2.fill(new Rectangle2D.Double(20, 0, 20, 20));

        Graphics2D g2 = (Graphics2D) g;
        g2.setPaint(new TexturePaint(bi, new Rectangle(40, 40)));
        g2.fill(new Rectangle2D.Double(x+10, y+10, 380, 85));
        g2.setStroke(new BasicStroke(20));
        g2.draw(new Rectangle2D.Double(x+20, y+115, 360, 65));
        g2.setFont(new Font("SansSerif", Font.BOLD, 24));
        g2.drawString("Text using TexturePaint", x+40, y+155);
    }
```

```
    private void image(Graphics g, int x, int y) {
        BufferedImage bi = new BufferedImage(40, 40, BufferedImage.TYPE_INT_RGB);
        Graphics2D bi2 = (Graphics2D) bi.getGraphics();
        // Rita en bild
        ImageIcon ii = new ImageIcon("images/Dylan.jpg");
        bi2.drawImage(ii.getImage(), 0, 0, 40, 40, null);

        Graphics2D g2 = (Graphics2D) g;
        g2.setPaint(new TexturePaint(bi, new Rectangle(40, 40)));
        g2.fill(new Rectangle2D.Double(x+10, y+205, 380, 90));
        g2.setStroke(new BasicStroke(20));
        g2.draw(new Rectangle2D.Double(x+20, y+315, 360, 65));
        g2.setFont(new Font("SansSerif", Font.BOLD + Font.ITALIC, 24));
        g2.drawString("Text using TexturePaint", x+40, y+355);
    }

    public void paintIcon(Component c, Graphics g, int x, int y) {
        texture(g, x, y);
        image(g, x, y);
    }

    public int getIconWidth() {
        return 400;
    }

    public int getIconHeight() {
        return 400;
    }

    public static void main(String[] args) {
        IconWindow.showIcon(new Exercise10());
    }
}
```