

# Laboration 1: *Figurer i hierarki och JUnit*

## Bakgrund

Två grundläggande tekniker i objektorienterad konstruktion är *arv* och *komposition*. Vidare, då man skriver kod, har det visat sig att det är bra att börja med att skriva tester som kan köras automatiskt. Dessa tester körs sedan om och om igen vid minsta lilla förändring, integration eller liknande för att visa (övertyga om) att koden (fortfarande) gör vad som är tänkt.

## Mål

Laborationens främsta mål är att ge övning i att organisera och strukturera program med hjälp av arvshierarkier och komposition.

Ett annat mål är att du ska utveckla en vana att skriva tester för och att kontinuerligt enhetstesta din kod.

## Uppskattad tidsåtgång

18 timmar per gruppmedlem.

## Deadline

28 september.

## Förberedelse/genomförande

Om ni inte redan kan eller kommer ihåg Java börja med att lära er grunderna i Java genom att:

- Genomföra början av <http://java.sun.com/docs/books/tutorial/index.html> som innehåller en bra introduktion och enkel Java-kurs.

Fortsätt sedan med att genomföra labben genom att:

- Läs uppgiftslydelsen.
- Repetera lämpliga delar av föreläsninganteckningarna <http://www.nada.kth.se/kurser/kth/2D1359/01-02/contents/forelasningar/lecturenotes.html>
- Läs lämpliga delar av dokumentationen av JUnit och pröva paketet. Se också beskrivningen av hur du kan köra JUnit på NADAs SUNar och kodexemplen från föreläsning 2 på sidan med föreläsninganteckningar.
- Pröva genoma att först skriva tester för och konstruera delar av någon av figurklasserna (ni kan komplettera och/eller modifiera denna klass senare, så allt behöver inte vara med från början).
- Gör resten av labben.

## REGLER, HJÄLP OCH ANVISNINGAR

Följ också reglerna för hur laborationerna skall genomföras med bla kodstandard, testkod som skrivs innan implementation som sedan skall fungera efter varje förändring.

Använd en webbläsare och läs vad som står på följande två websidor:

<http://java.sun.com/docs/codeconv/index.html>, för kodkonventioner som ni ska försöka följa

<http://www.nada.kth.se/kurser/kth/2D1359/01-02/contents/forelasningar/lecturenotes.html#F2>  
(se alla kolumner!) för referenser till bland annat JUnit, XP och exempel

## Tips

- "Kursen" på <http://java.sun.com/docs/books/tutorial/index.html>.
- Klassen `Vector` kan vara användbar vid komposition.

## Uppgifter

Ni skall skriva ett program i vilket man kan skapa några olika typer av grafiska figurer.

I beskrivningen av klasser, attribut och metoder som följer är inga modifierare (`public`, `protected`, `private`, `final` osv) angivna. Ni får själva välja dessa på lämpligt sätt. Vidare är ni fria att använda andra attribut eller egna metoder (till exempel hjälpmetoder) om ni kan motivera detta.

Testa också all funktionalitet genom att använda JUnit för att skriva enhetstester för klasserna. Följ också XP:s "regel" att skriva testerna *innan* ni skriver produktionskoden.

Förbättra också kontinuerligt kodens struktur. Det vill säga har ni dåligt valda namn på metoder, attribut eller klasser så ändrar ni dem till bättre, har ni långa metoder så delar ni upp dem på flera, använder svåräst/svårförstålig kod så skriver ni om den, osv.

### Klasser

Konstruera klasser för att hantera följande typer av *grafiska figurer*:

- rektangel
- kvadrat
- cirkel
- ellips
- polygontåg
- polygon.

Ordna klasserna i en lämplig *hierarki*.

Konstruera en *abstrakt klass* som alla andra klasser ärver ifrån.

### Attribut

Varje objekt skall ha ett *namn* som anges vid skapandet av objektet.

Ni måste också bestämma om ni behöver attribut för objektens *position* och *utsträckning* eller kanske för dess *tjocklek*, *färg*, mm. Vissa attribut behövs för vissa typer av objekt men inte för andra, till exempel en grupp av figurer (beskrivs nedan) behöver hantera *medlemmar*, en polygon en lista med dess *punkter*.

Placera attribut som är gemensamma för alla klasser i den abstrakta klassen (till exempel attributet `name`).

### Instansmetoder

Följande metoder skall implementeras. För varje metod är typ på eventuellt resultat och argument angivna. Namnet `Figur` är använt som namn på den abstrakta klassen.

Instansmetoder för *alla klasser*, dvs metoder för meddelanden som skall ha mening för alla figurer:

- `String getName()`, vilket namn har objektet (givet vid instansieringen)?
- `String toString()`, som ger en textuell presentation av objektet, anropas bla av "print-metoder" (`System.out.print(objekt);`). Tex kan man i kvadratens version av metoden definiera att en den skall skrivas på följande sätt:  
"Kvadrat position: (10, 20) utsträckning: 100".
- `Point getLocation()`, objektets position tagen i någon referenspunkt. Kan vara framräknad för till exempel polygoner (och senare grupper).
- `void setLocation(Point)`, flytta objektet till viss position. Hitta också på något lämpligt förfarande för polygoner (och senare grupper).
- `void moveBy(Point)`, flytta objektet en viss sträcka
- `Rectangle getBounds()`, den minsta rektangel som omsluter hela objektet.
- `boolean containsPoint(Point)`, befinner sig den givna punkten inom objektets omslutande rektangel eller ej?
- `boolean intersects(Figur)`, skär dom omslutande rektanglarna för mottagaren och argumentet varandra?
- `float distanceTo(Figur)`, hur långt ifrån varandra befinner sig mottagaren av meddelandet och objektet som är givet som argument. För enkelhets skull kan avståndet mellan objektens "location()" användas.

Instansmetoder för *polygon och polygontåg*:

- `void addPoint(Point)`, för att lägga till en ny punkt sist i figuren.

Metoder som kan definieras i den abstrakta klassen skall definieras där. Metoder som måste implementeras i dom konkreta subclasserna skall deklarerars som *abstrakta* i den abstrakta klassen.

### Konstruktörer

Alla figurer skall skapas med åtminstone angivande av ett *namn*. Det skall inte att gå att skapa ett objekt utan att ange ett namn.

Då en *rektangel, kvadrat, ellips och cirkel* skapas skall position och utsträckning anges. Inget annat sätt är tillåtet för dessa objekt. Utsträckningen för en rektangel och ellips ges som en punkt och för en kvadrat respektive cirkel som ett heltal. Till exempel

```
new MinEllips("Ellips 1", //namn
             new Point(10, 10), //position
             new Point(50, 100)); //utsträckning

new MinKvadrat("Kvadrat 1", //namn
              new Point(15, 15), //position
              25); //utsträckning
```

För *polygontåg och polygoner* skall det finnas konstruktörer där man kan skapa en av dem *utan att de innehåller några punkter från början, med en punkt, två punkter, tre punkter eller en vektor av punkter*. I alla fallen skall också (som vanligt) ett namn anges. Punkter skall sedan kunna läggas till efter instansieringen genom att metoden `addPoint` används.

### Grupp

Konstruera en klass för hantering av grupper av dom andra objekten (dvs av instanser av subclasser till den abstrakta figurklassen). Gruppens klass skall också vara subclass till den

abstrakta figurklassen (och därmed kunna ingå som element i en grupp). Alla dom attribut och metoder som skulle finnas för alla objekt skall också på lämpligt sätt implementeras för gruppen.

En grupp skall kunna skapas på fyra olika sätt (dvs det skall finnas konstruktörer för dessa situationer):

1. Genom att ge den ett namn, utan att ange några medlemmar  

```
new MinGrupp("Grupp 1");
```
2. Med namn och två andra figurer  

```
new MinGrupp("Grupp 2",
             rektangel, //rektangel och cirkel
             cirkel); //är variabler
```
3. Med namn och en lista av medlemmar där listan är en Vector  

```
new MinGrupp("Grupp 3",
             grupp2.members());
```
4. Med namn och skapad från en annan grupp  

```
new MinGrupp("Grupp 4",
             grupp3);
```

Skapa också följande instansmetoder:

- `void addMember(Figur)`, för att lägga till ett element sist i gruppen.
- `Vector getMembers()`, en lista av alla gruppens medlemmar.

### Effektivisering

Gör följande förbättring:

- Förbättra `containsPoint(Point)` för ellips och cirkel så att du använder figurens rand istället för den omskrivna rektangeln. Du kan utnyttja ellipsens ekvation:

$$\frac{(x - m_x)^2}{a^2} + \frac{(y - m_y)^2}{b^2} = 1$$

där  $(x, y)$  är en punkt på ellipsen,  $(m_x, m_y)$  är ellipsens mittpunkt,  $a$  är halvaxelns längd i x-riktningen och  $b$  är halvaxelns längd i y-riktningen. För punkter inuti ellipsen är vänsterledet  $< 1$ .

### Redovisning

Ni ska kunna visa upp snygg fungerande kod, kunna redogöra för dess struktur och motivera varför ni gjort på ett visst sätt.

Att koden fungerar visar ni genom att visa handledaren och köra dom automatiska tester som ni konstruerat samt tester som visar att följande fungerar (om ni nu inte redan har dessa tester sedan tidigare):

- skapa namngivna instanser av alla olika figurtyper.
- gruppera namngivna objekt.
- avgruppera en grupp.
- fråga om två objekt skär varandra.
- flytta ett objekt.
- fråga om avståndet mellan två objekt.

Observera att ni måste visa att det hela fungerar för alla typer av figurer som rektanglar, polygoner, cirklar och grupper (eller grupper av grupper) samt, i dom situationer där detta krävs, kombinationer av olika typer av figurer.