# TENTAMEN: Objektorienterade applikationer

## Läs detta!

- *Uppgifterna är inte avsiktligt ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad.
- Skriv din tentamenskod på varje blad (så att vi inte slarvar bort dem).
- **Skriv rent dina svar. Oläsliga svar r ä t t a s e j!**
- Programmen skall skrivas i Java 5 eller senare version, vara indenterade och renskrivna, och i övrigt utformade enligt de principer som lärts ut i kursen.
- Onödigt komplicerade lösningar ger poängavdrag.
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Givna deklarationer, parameterlistor, etc. får ej ändras, såvida inte annat sägs i uppgiften.
- Läs igenom tentamenstesen och förbered ev. frågor.

| |
|---|
| I en uppgift som består av flera delar får du använda dig av funktioner klasser etc. från tidigare deluppgifter, även om du inte löst dessa. |

*Lycka till!*

## Uppgift 1

Rita ett UML-diagram som motsvarar följande relationer mellan klasserna A, B, C och D:

- en B är en A
- en C är en A
- en D är en A
- en D kan ha 0 eller flera A

Relationerna utgör tillsammans ett exempel på ett känt designmönster, vilket?

(6 p)

## Uppgift 2

Konstruera en klass med namnet `DieButton` som subklass till `JButton`. När man trycker på en `DieButton` i fönstret skall tärningen visa ett nytt slumpmässigt värde. Varje objekt av klassen skall fungera som en självständig tärning. Ex. Ett fönster med tre tärningar:


För att sätta lämpliga bilder på tärningen beroende på vilket tärningsvärde som skall visas finns till din hjälp den färdiga klassen

```
public class DieIcons {
    public static DieIcons getInstance() { ... }
    public int size() { ... }
    public ImageIcon getIcon(int i)
        throws IndexOutOfBoundsException { ... }
}
```

Observera att klassen konstruerats enligt Singleton-mönstret. Den hämtar automatiskt ikonerna från bildfiler. Vi går inte in närmare på hur klassen är implementerad.

- `getInstance` ger tillgång till ett objekt av klassen.
- `size` returnerar antalet olika ikoner som objektet har.
- `getIcon(i)` returnerar en ikon med `i` ögon.
  Tärningen skall ha lika många sidor som det finns ikoner.

(8 p)

## Uppgift 3

En viss mätutrustning sparar värden av typen **`double`** i en binärfil. En server skall kunna skicka dessa värden till klienter.

a) Konstruera ett serverprogram som lyssnar efter klientanslutningar. Värdena i binärfilen skall skickas till varje ansluten klient enligt client-server-metoden. Programmet skall ha en `main`-metod och man skall kunna ge serverns portnummer och namnet på indatafilen med mätvärdena som programargument. Ex. `> java server 1234 values.dat`

(12 p)

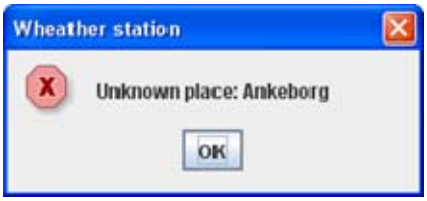
b) Konstruera ett klientprogram som kopplar upp sig mot en server enligt a och hämtar mätvärdena. Varje värde skall konverteras till text och skrivas till en textfil, en rad per värde. Programmet skall ha en `main`-metod och man skall kunna ge serverns IP-adress, dess port, samt namnet på utdatafilen som programargument.
Ex. `> java client 127.0.0.1 1234 values.txt`

(10 p)

## Uppgift 4

Ett grafiskt användargränssnitt för en liten väderstation skall ha följande utseende:


Man skall kunna skriva in namnet på en ort i det vänstra textfältet. När man trycker på en av knapparna skall motsvarande väderparameter visas i textfältet till höger. Om man anger en ort som inte existerar (?) så skall ett dialogfönster visas:




Till din hjälp finns den färdiga klassen

```
public class Database extends Observable {
    public void computeTemperature(String place)
        throws IllegalArgumentException { ... }

    public void computeWindSpeed(String place)
        throws IllegalArgumentException { ... }

    public void computePressure(String place)
        throws IllegalArgumentException { ... }
}
```

Metoderna i `Database` hämtar och lagrar väderdata för angiven plats. Om platsen ej kan hittas kastas undantaget `IllegalArgumentException`. Om platsen hittas uppdateras alla observatörsobjekt med efterfrågad väderparameter i form av en sträng, t.ex. "10.8 gr C". Vi går inte in på hur `Database` är implementerad. Uppgiften är att konstruera den grafiska gränssnittsklassen `Gui`. Klassen skall baseras på designmönstret `Observer`. Eftersom de tre väderparametrarna hanteras på likformigt sätt räcker det att du skriver kod för hantering av temperaturdata. Skriv alltså kod som motsvarar knappen längst till vänster och de två textfälten.

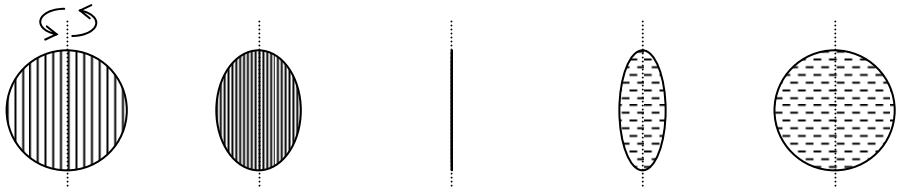
(14 p)

## Uppgift 5

Antag att vi har klasser för plana geometriska figurer som t.ex. cirklar och kvadrater. Figurerna kan ritas upp och raderas. Figurklasserna implementerar gränssnittet

```
public interface Drawable {
    void draw(int rotationAngle);
    void erase();
}
```

Uppritningen är lite speciell eftersom man kan bestämma i vilken rotationsvinkel figuren skall ritas. Rotationen sker längs figurens vertikala centrumlinje. T.ex. ser en cirkelskiva med olika mönster på fram- och baksidan ut ungefär så här om man ritar den med rotationsvinklarna 0, 45, 90, 135 resp. 180 grader.



Givet sådana figurklasser kan man skapa en klass för att animera roterande figurer. Klassen `Animation` skall bl.a. ha metoden

```
/**
  * horizontalSpin rotates the shape around its vertical
  * center axis. The animation is performed by painting
  * the shape in 60 different rotation angles evenly distributed
  * around a complete 360 degree cycle.
  *
  * @param  revolutions   the number of complete revolutions
  * @param  rpm   the rotation speed in revolutions per minute
  */
 public void horizontalSpin(int revolutions,int rpm)
```

Implementera klassen i enlighet med designmönstret Decorator. Om man t.ex. vill rotera en kvadrat tio varv med ett varv per sekund, och därefter ta bort den, skall man kunna skriva

```
Animation sq = new Animation(new Square(...));
sq.horizontalSpin(10,60);
...
sq.erase();
```

Tips: Tidsfördröjning kan fås med metoden **`static void`** `Thread.sleep()`.

(10 p)