Lösningsförslag till april omtentamen för TDA551 Objekt-orienterad Programmering och Design

Dag: 2017-04-10, *Tid:* 08.30-12.30

Uppgift 1

a) En konstruktor är del av en klass och skapar objekt av den klassen. En konstruktor liknar en metod och brukar initialisera instansvariabler.

b) En typ är en subtyp till en annan typ, om alla element (värden eller objekt) av subtypen kan användas som om de vore av supertypen. En subtyp har (åtminstone) alla element (metoder, variabler) som supertypen också har, och tillåter subtyp-polymorfism.

c) En generisk typ är en typ som är parametriserad över en (eller flera) typ.

d) Ett gränssnitt är en samling metod-signaturer (dvs bara returtyp, metodnamn och parameterlista) och kan implementeras av en klass.

Uppgift 2

a) Korrekt: `public String getDescription()` i `Tower` klassen blir anropad. Metoden `getDescription()` i `Piece` klassen blir inte anropad, den är ju abstrakt.

b) Inkorrekt och ger en statiskt typ fel: `Piece` är abstrakt och man kan inte skapa objekt av abstrakta klasser.

c) Korrekt: `public String getDescription()` i `Tower` klassen blir anropad.

d) Korrekt: först blir `public String getDescription()` i `PowerTower` klassen anropad och sedan `public String getDescription()` i `Tower` klassen.

Uppgift 3

a) i) Korrekt.

ii) Inkorrekt: `Tower orthanc = listExtendsPiece.get(0)` ger ett typfel. Vi kan bara läsa element med typen `Piece` (eller dess supertyper).

iii) Inkorrekt: `listPiece = listTower` ger ett typfel. En generisk lista är invariant i sin typ-parameter.

iv) Inkorrekt: `Piece piece = listSuperTower.get(0)` ger ett typfel. Vi kan bara läsa element av typen `Object` för `listSuperTower` är contravariant i sin typ-parameter.

b) `listPiece`

c) Vad `?` än är så vet vi att den är en supertyp till `T` och det innebär att vi kan inte läsa från listan (förutom objekter av typen `Object`) men vi kan lägga till objekt av typen `T`. Till exempel, vi kan lägga till objekt av typen `Tower` i listor med typen `List<? super Tower>` så som `List<Piece> listPiece`.

Uppgift 4

a) Med en 'race condition' kan vi hamna i ett ogiltigt tillstånd om flera trådar 'tävlar' om samma resurser (variabler). Till exempel, om vi har två trådar som samtidigt vill göra ändringar i ett bankkonto, då kan vi hamna i ett ogiltigt tillstånd.

b) Med modifieraren `synchronized` kan vi markera en metod (eller sats) som en kritisk sektion. En sådan metod bara körs av *en* tråd i taget, som låser sektionen. En annan tråd som vill också köra metoden får vänta.

Om man använder modifieraren för mycket (dvs på metoder som inte har behov) då får man dålig paralellism. Dessutom kan vi få 'deadlock' problem.

Uppgift 5

a) 1. anropa `addTower`
   2. anropa `getCanvas` och sedan använda referensen för att ändra `canvas` instansvariabeln (dvs 'aliasing')
   3. samma för `getMonster`
   4. samma för `getTowers`

b) Man kan använda sig av *defensive copying* för att förhindra 'aliasing' problem. Till exempel, man kan skriva om gettern för monstret så här:

```
public Monster getMonster() {
  return new Monster(monster);
}
```

och lägga till en sådan konstruktor i `Monster` klassen.

Uppgift 6

a) Liskov Substitution principen säger att en typ `S` är en äkta subtyp till `T` endast om, för varje publik metod som finns i både `S` och `T`: `S`'s metod godtar alla värden som `T`'s metod godtar och att `S` gör alla beräkningar på denna indata som `T` gör (och kanske fler). Dvs att `S` beter sig som en `T` och vi blir inte överraskad när vi anroper dess metoder.

Ett klassiskt exempel är det `Square` och `Rectangle` problem: en `Square` ärver från `Rectangle` men alla metoder kan inte göra samma beräkningar.

```
class Rectangle {
  private int width;
  private int height;

  public void setSize(int x, int y) {
    width  = x;
    height = y;
  }
}

class Square extends Rectangle {
  public void setSize(int x, int y) {
    // vad ska man göra här?
  }
}
```

b) Command-Query Separation principen säger att en metod ska antingen ha sido-effecter (en *mutator*) eller returnera ett värde (en *accessor*), inte båda. En metod som returnerar information om ett objekt ska inte ändra tillståndet hos objektet. En metod som ändrar tillståndet hos ett objekt ska (som regel) inte samtidigt returnera information om objektet. T.ex. metoden `sizeArea` i nedanstående klass bryter mot den här principen.

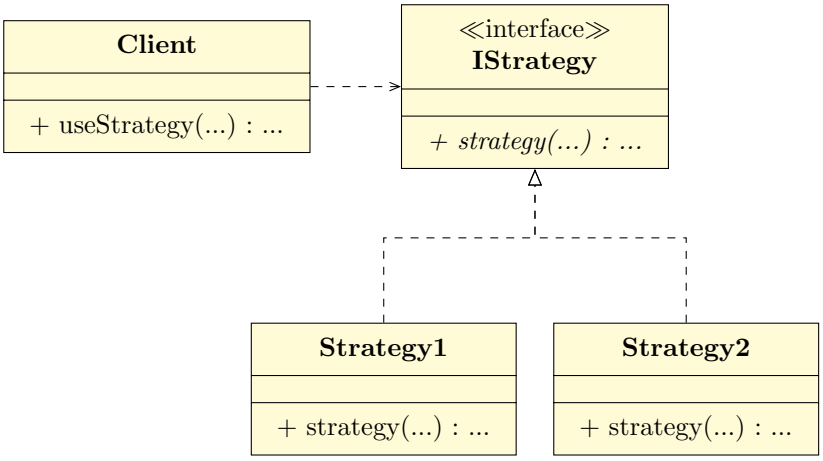
```
class Rectangle {
  private int width;
  private int height;

  public int sizeArea(int x, int y) {
    width  = x;
    height = y;
    return x * y;
  }
}
```

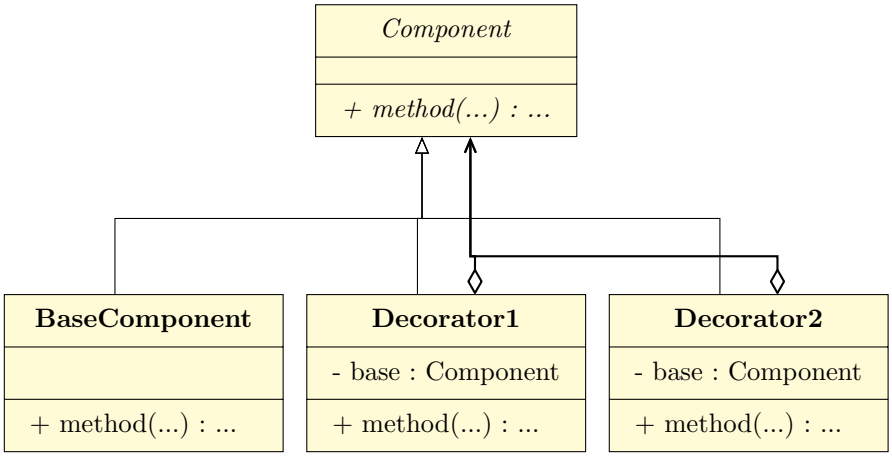
Man borde ha en separat metod för att ändra storleken och en för att beräkna arean.

Uppgift 7

a) En Strategy Pattern abstraherar en algoritm och tillåter att variera denna utan att påverka de som är beroende av det. Vi definierar ett fristående interface och kan sedan definiera helt olika beteenden som fristående klasser som implementerar detta interface.

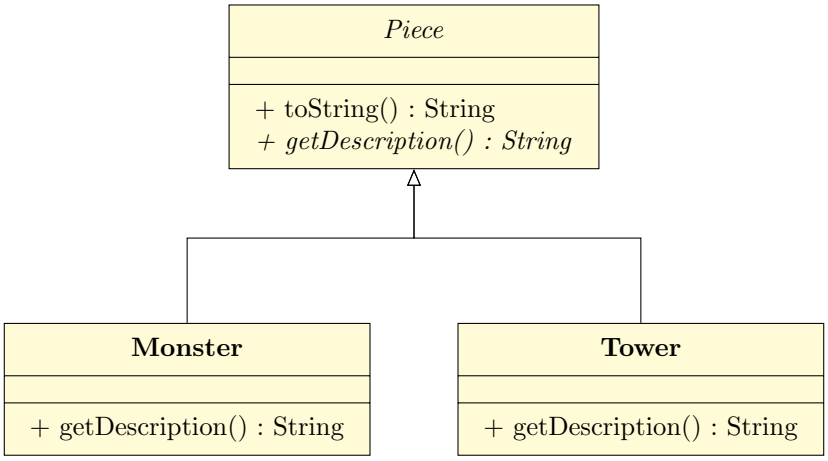


b) Genom att dela upp ett komplext objekt i flera beståndsdelar, som kan sättas ihop lager på lager med samma gränssnitt, så kan vi uppfylla Single Responsibility principen. Med Decorator Pattern representeras ett objekt i domänen vi modellerar av en sammansättning av objekt.



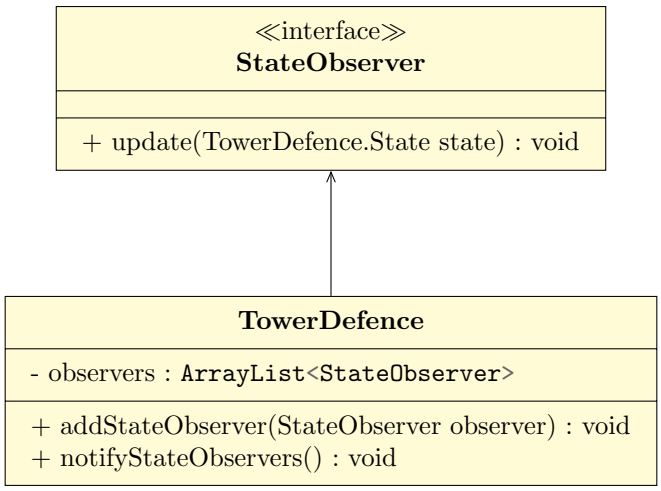
Uppgift 8

Template Method Pattern

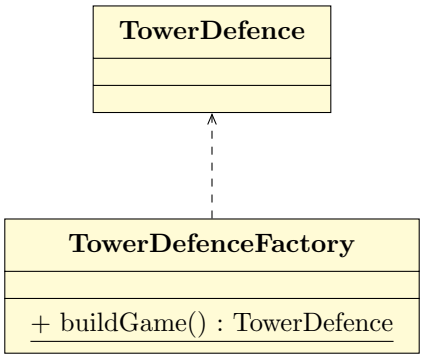


Observer Pattern

Vi har inte någon konkret observer men det spelar ingen roll, kärnan i Observer Pattern är att möjliggöra för sådana att existera.

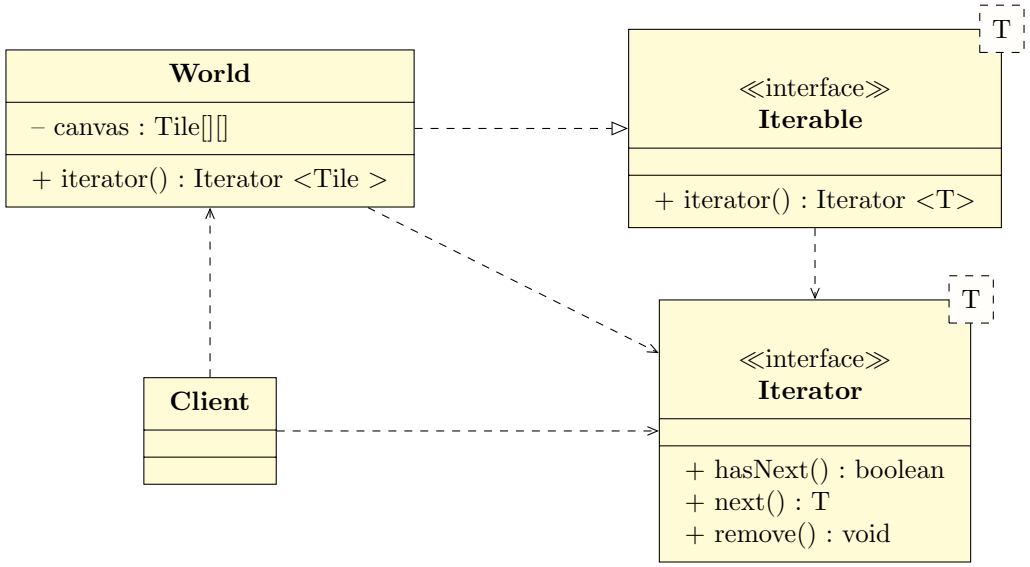


Factory



Uppgift 9

Se koden lite längre ner.



Koden

```
import java.util.*;

public class World implements Iterable<World.Tile> {
  // ...

  private Tile[][] canvas;

  // ...

  @Override
  public Iterator<Tile> iterator() {
    return new Iterator<Tile>() {
      private int index = 0;

      public boolean hasNext() {
        return index < rows * cols;
      }

      public Tile next() {
        int i = index / cols;
        int j = index % cols;
        index++;
        return canvas[i][j];
      }

      public void remove() {
        throw new UnsupportedOperationException("Not supported.");
      }
    };
  }
}
```

Uppgift 10

a)
- `TowerDefence` rad 8 (beroende på `ArrayList` isf `List`)
- `World` rad 6 (beroende på `ArrayList` isf `List`)

b)
- `TowerDefence` rad 17, 25, 30 (anropa `Monsters` metoderna)