TENTAMEN: Objektorienterad programmering

Läs detta!

- *Uppgifterna är inte ordnade efter svårighetsgrad.*
- Börja varje uppgift på ett nytt blad (utom i uppg. 1).
- Numrera och ordna bladen i uppgiftsordning.
- 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!**
- Programkod som finns i tentamenstesen behöver ej upprepas.
- Programkod skall skrivas i Java 5 (eller senare) och vara indenterad och snyggt uppställd.
- Du behöver ej skriva importdeklarationer.
- Onödigt komplicerade lösningar ger poängavdrag.
- Omotiverad användning av klassvariabler och klassmetoder ger poängavdrag.
- Givna deklarationer, parameterlistor etc. får ej ändras.
- 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

Välj **ett** alternativ för varje fråga! Garderingar ger noll poäng. Inga motiveringar krävs. Varje korrekt svar ger 2 poäng.

1. Följande klass används för att lagra uppgifter om personer

```
class Person {
    ...
    public void setAddress(String address);
    ...
}
```

Antag att `setAddress` ej kastar några undantag. Vilket omdöme är då korrekt om metoden

```
void changeAddress(Map<String,Person> m,String id,int newAddr) {
    m.get(id).setAddress(newAddr);
}
```

a. metoden kan ej orsaka att undantag kastas
b. metoden är ej typkorrekt
c. metoden kan orsaka att undantag kastas

2. Klasserna `A` och `B` definieras

```
public class A {
    public void f(long x) { g(x); }
    public void g(long x) { System.out.println("A.g"); }
    public void h(int x) { System.out.println("A.h"); }
}
public class B extends A {
    public void g(int x) { System.out.println("B.g"); }
    public void h(int x) { f(x); }
}
```

Vad skrivs ut av följande kod?

```
A obj = new B();
obj.h(42);
```

a. `A.g`
b. `A.h`
c. `B.g`

3. Vilket påstående om en objektorienterad design stämmer bäst?
   a. Låg kohesion medför ofta hög koppling.
   b. Hög kohesion medför ofta hög koppling.
   c. Låg kohesion medför ofta låg koppling.

4. Givet

```
public interface A
public abstract class B implements A
public class C extends B
```

Vilket är typkorrekt?

a.
```
B x = new B();
A y = x;
```

b.
```
A x = new C();
B y = x;
```

c.
```
B y = new C();
A x = y;
```

d. Inget av ovanstående.

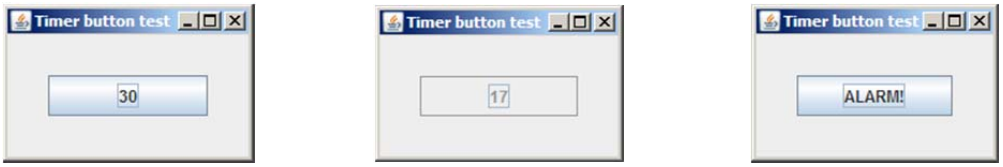
5. En av överskuggningarna i `B` är otillåten i Java, vilken?

```
public class A {
    public void f() {}
    protected void g() {}
    private void h() {}
}
public class B extends A {
    protected void f() {}     // a
    public void g() {}        // b
    public void h() {}        // c
}
```

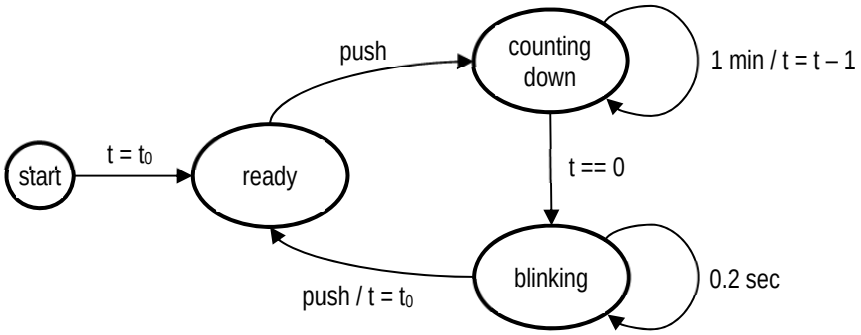
(10 p)

Uppgift 2

Många av oss sitter still alldeles för länge framför datorn och behöver en påminnelse om att röra på oss ibland! Konstruera därför en liten GUI-komponent som larmar när man suttit still för länge genom att definiera en klass med namnet `TimerButton` som subklass till `JButton`. Först skall knappen visa en förinställd tid t_0, t.ex. 30 minuter. När man trycker på knappen skall tiden som visas börja räkna ner med en minuts intervall. När tiden når noll skall knappen börja blinka genom att omväxlande visa texten ”ALARM!” resp. ingen text med 0.2 sekunders intervall. Om man nu trycker på knappen skall blinkningen avbrytas och knappen återgå till ursprungsläget. Ex.



Följande tillståndsdiagram sammanfattar komponentens beteende:



Klassen skall ha en konstruktor som tar den önskade tiden i minuter som inparameter.
I uppgiften ingår enbart att konstruera klassen `TimerButton`.

(10 p)

Uppgift 3

Punkter i planet kan representeras med klassen:

```
public class Point {
    private int x,y;
    public Point(int x,int y) {
        this.x = x;
        this.y = y;
    }
    public long getX() { return x; }
    public long getY() { return y; }
}
```

Vidare finns följande klass för representation av polygoner:

```
public class Polygon implements Cloneable {
    private ArrayList<Point> points;
    private int noOfPoints;
    ...
}
```

a) Skriv en `equals`-metod för `Point`.

(3 p)

b) Skriv en `hashCode`-metod för `Point`.

(1 p)

c) Skriv en `clone`-metod för `Point`. Metoden skall returnera en djup kopia.

(2 p)

d) Skriv en `clone`-metod för `Polygon`. Metoden skall returnera en djup kopia.

(4 p)

Uppgift 4

En viss stad i ett nordeuropeiskt land har infört biltullar och samlar in statistik över antalet fordonspassager per timma genom de olika betalstationerna. Efter varje hel timma rapporterar stationerna antalet passager till en server. Alla inkomna data skrivs till en *binärfil*. För varje timobservation lagras följande, i nämnd ordning: Stationens id som ett 32 bitars heltal (`integer`), tidpunkten som ett timnummer i intervallet [1,24] som ett 32 bitars heltal, samt antalet fordonspassager som ett 64 bitars heltal (`long`). I filen lagras ett antal sådana observationer i en enda följd: $id_1t_1\#_1id_2t_2\#_2\ldots$ Varje fil innehåller observationer för ett dygn. Timma 1 avser fordonspassager mellan kl. 00.00 och 01.00, etc.

Uppgiften är att konstruera en klass `TrafficData` som kan läsa en sådan binärfil och ge användaren möjlighet att enkelt få reda på antalet fordonspassager under en given timma genom en viss betalstation. Klassen skall ha följande konstruktor och metoder:

- en konstruktor som tar namnet på datafilen som inparameter, öppnar filen, samt läser in alla data till en tabell som lagras i en instansvariabel.
- `public boolean hasStation(int stationId)`
  returnerar **`true`** om en betalstation med angivet id finns, annars **`false`**.

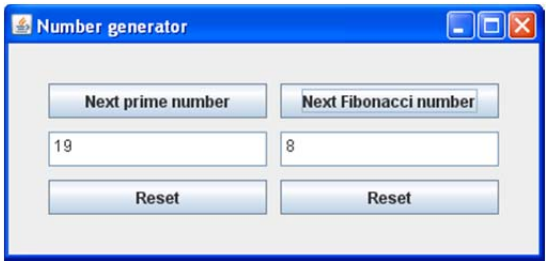
- `public long getNoOfVehicles(int stationId,int hour)`
  `throws IllegalArgumentException`
  returnerar antalet fordonspassager för angiven station och timma. Kastar `IllegalArgumentException` om `hour` ej anger en giltig timma i intervallet [1,24]

Representera tabellen med en lämplig standardklass. Använd ett fält per betalstation för att lagra de olika timvärdena. Du kan anta att stationerna skickat in data för dygnets samtliga timmar. Inga antaganden får dock göras om antalet stationer, eller i vilken ordning timobservationerna ligger i filen. Läs data från filen med lämplig standardklass. Filslut hanteras genom att fånga undantaget `EOFException`.

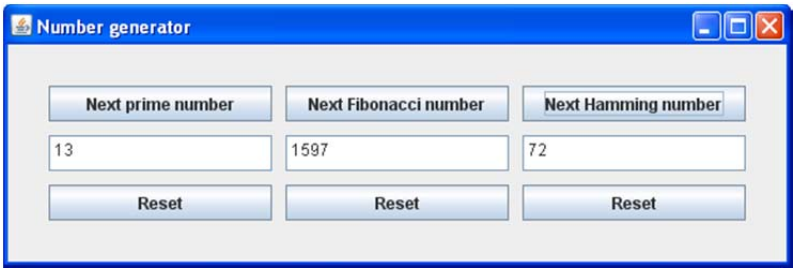
(10 p)

Uppgift 5

I kursen har vi stött på ett program som låter användaren studera olika talserier. Programmet visar ett fönster på skärmen:



Vi skall bygga ut programmet så att man även kan skapa s.k. *hammingtal*. De kan skrivas som produkter av potenser av talen 2, 3 och 5, d.v.s. $\{ 2^a \cdot 3^b \cdot 5^c \mid a, b, c \geq 0 \}$. De minsta hammingtalen är 1, 2, 3, 4, 5, 6, 8, 9, 10, 12, 15, … Efter utbyggnaden skall fönstret alltså ha en tredje kolumn för hammingtal



Programmet är strukturerat enligt MVC-arkitekturen. Varje talserie beräknas i en subklass till följande klass

```
public abstract class NumberGenerator extends Observable {
    public abstract void computeNext();
    public abstract void reset();
}
```

Första uppgiften är att implementera subklassen `HammingGenerator`, den andra att komplettera det grafiska gränssnittet.

a)
Implementera klassen `HammingGenerator` som subklass till `NumberGenerator`. Metoden `reset` skall återställa aktuellt hammingtal till 1 och uppdatera observatörerna med talet. Metoden `computeNext` skall beräkna nästa tal i serien och uppdatera observatörerna med talet. Talen skall överföras som siffersträngar. Tips: om du lagrar talen i en prioritetskö (se API) så kommer de att tas ut ur kön i storleksordning. Börja med att stoppa in 1 i kön. För varje tal som tas ut läggs produkter av talet och lämpliga faktorer tillbaka i kön. Processen skapar en del duplikat, vilka skall ignoreras, så att bara unika tal rapporteras till observatörerna.

(6 p)

b)
Följande vyklass innehåller den del av koden som behöver förändras. Ange vad som behöver ändras.

```
public class UserInterface extends JFrame {
    public UserInterface(PrimeGenerator primeGen,
                         FibonacciGenerator fibGen)
    {
        setTitle("Number generator");
        setLayout(new GridLayout(1,2,10,10));

        JPanel primePanel =
            new NumberPane("Next prime number",
                           new NextButtonController(primeGen),
                           new ResetButtonController(primeGen));
        primeGen.addObserver((Observer)primePanel);
        add(primePanel);

        JPanel fibPanel =
            new NumberPane("Next Fibonacci number",
                           new NextButtonController(fibGen),
                           new ResetButtonController(fibGen));
        fibGen.addObserver((Observer)fibPanel);
        add(fibPanel);

        pack();
        setVisible(true);
    }
}
```

(2 p)

c)
Vilka ändringar behöver göras i `main`?

```
public class Main {
    public static void main(String[] args) {
        PrimeGenerator primeGen = new PrimeGenerator();
        FibonacciGenerator fiboGen = new FibonacciGenerator();
        new UserInterface(primeGen,fiboGen);
        primeGen.reset();
        fiboGen.reset();
    }
}
```

(2 p)

Uppgift 6

Iteratorer i Javas API implementerar som bekant gränssnittet

```
public interface Iterator<T> {
    boolean hasNext();
    T next();
    void remove();
}
```

Det skulle vara användbart om iteratorer också hade en metod för att söka upp ett element som har en viss egenskap:

```
public T seek(Property<T> p)
```

`M`etoden skall avancera iteratorn tills ett element som har egenskapen som beskrivs av `p` hittas. Om ett sådant element hittas skall en referens till det returneras, annars `null`. Om det funna elementet var det sista i samlingen, eller om inget element hittades, skall `hasNext` returnera **`false`**. Annars skall iteratorn placeras på elementet efter det funna. Gränssnittet `Property` ser ut på följande sätt

```
public interface Property<T> {
    boolean hasProperty(T x);
}
```

Vi kan t.ex. skapa en klass för att testa om ett tal är positivt:

```
public class IsPositive implements Property<Integer> {
    public boolean hasProperty(Integer x) {
        return x > 0;
    }
}
```

Eftersom iteratorkklasserna inte är åtkomliga för arv skall du tillämpa designmönstret *Decorator* och definiera klassen `SeekIterator` enligt mönstret så att man kan skriva t.ex.

```
ArrayList<Integer> l = new ArrayList<Integer>();
for ( int i = -10; i < 10; i++ )
    l.add(i);

SeekIterator<Integer> it = new SeekIterator<Integer>(l.iterator());

Integer firstPositive = it.seek(new IsPositive());
if (firstPositive != null ) {
    System.out.print(firstPositive);    // 1 skrivs ut
    while ( it.hasNext() )
        System.out.print(it.next());    // 2 3 4 ...
}
```

(10 p)