Övning vecka 5.

Denna vecka ska vi titta pa samlingar, generics och designmönstren Decorator, Singleton och Iterator.

Uppgift 1

Kom ihåg att samlingar och avbildningar (eng. "map") endast kan lagra objektreferenser. Om du försöker lagra värden av en primär datatyp, kommer Javakompilatorn automatiskt göra omslagsobjekt (wrapper objects) av motsvarande typer, d.v.s. automatisk typomvandling. (Om du inte är säker på vad omslagsobjekt är för något, diskutera detta med varandra och/eller handledaren).

a) Skriv ett program som genererar en frekvenstabell av orden i argumentlistan till programmet. Frekvenstabellen mappar varje ord till antalet gånger det förekommer i argumentlistan. För att förstå vad som egentligen händer, får ni för närvarande inte använda er av automatisk typomvandling (boxing, unboxing), utan ni måste göra detta explicit. Använd därför objekt av klassen IncrementableInteger nedan, som "värden" i avbildningen. Histogrammet skall alltså ha den statiska typen Map<String, IncrementableInteger>.

```
public class IncrementableInteger {
   private int intField;
   public IncrementableInteger(int i) {
       intField = i;
   }
   public void increment() {
       intField = intField + 1;
   }
   public int getIntField() {
       return intField;
   }
   public String toString() {
       return String.valueOf(intField);
   }
}//IncrementableInteger
```

b) Skriv nu samma program utan att använda IncrementableInteger. Använd automatisk typomvandling där ni kan, d.v.s. programmet ser ut att spara och hämta **int**:s. Deklarationen av avbildningen får nu den statisk typen Map<String, Integer>. Kan ni se några nackdelar med denna version?

Uppgift 2

Betrakta klassen DummyDates på nästa sida. Klassen skapar (en något meningslös) lista över slumpade datum. Klassen tillhandahåller även en iterator för att stega sig igenom de genererade datumen. Men iteratorn har brister, identifiera dessa! Till din hjälp har du följande utdrag ur Suns Java 6 API:

boolean hasNext()	Returns true if the iteration has more elements. (In other words, returns true if next would return an element rather than throwing an exception.)
E next()	Returns the next element in the iteration. Throws NoSuchElementException if the iteration has no more elements.
void remove()	Removes from the underlying collection the last element returned by the iterator (optional operation). This method can be called only once per call to next. The behavior of an iterator is unspecied if the underlying collection is modied while the iteration is in progress in any way other than by calling this method. Throws UnsupportedOperationException if the remove operation is not supported by this Iterator. Throws IllegalStateException if the next method has not yet been called, or the remove method has already been called after the last call to the next method.

```
public class DummyDates implements Iterable<Date> {
   private final static int SIZE = 15;
   private Date[ ] dates = new Date[SIZE];
   private int actualSize;
   public DummyDates() {
      Random r = new Random();
      // Fill the array. . . .
      for (int i = 0; i < SIZE; i++) {
         dates[i] = new Date(r.nextInt());
      }
      actualSize = SIZE;
   }
   public Iterator<Date> iterator() {
      return new DSIterator();
   }

   private class DSIterator implements Iterator<Date> {
      // Start stepping through the array from the beginning
      private int next = 0;
      public boolean hasNext() {
         // Check if a current element is the last in the array
         return (next < actualSize);
      }
      public Date next() {
         return dates[next++];
      }
      public void remove() {
         for (int i = next; i < SIZE - 1; i++) {
            dates[i] = dates[i + 1];
         }
         dates[SIZE - 1] = null;
         actualSize--;
      }
   }//DSIterator
}//DummyDates
```

Uppgift 3

Implementera metoden

```
public static <T> Set<T> exclusiveUnion(Set<T> s1, Set<T> s2) . . .
```

som returnerar mängd av element som antingen finns i mängden `s1` eller mängden `s2` (elementen får alltså inte förekomma både i `s1` och `s2`). Metoden får inte vara destruktiv, d.v.s. innehållet i `s1` och `s2` får inte förändras. I metodkroppen använder ni bara mängdoperationer och konstruktorer för mängder.

Uppgift 4

Nedan ges ett enkelt testprogram som lagrar NUMRUNS slumpmässigt genererade objekt av typen Integer i en samling och därefter efter generera ytterligare NUMRUNS slumpmässigt objekt av typen Integer samt söker efter dessa i samlingen.

Tre olika samlingar använts i testprogrammet: en trädstruktur, en hashtabell och en länkad lista. Körningstider (i sekunder) för att utföra add-operationerna och contains-operationerna beräknas för respektive samling. I tabellen nedan redovisas resultaten för för exekveringar av programmet då 100k respektive 5M slumptal genererades. Vardera av x, y, z motsvarar varsin samling (datastruktur). Vilken datastrukturer motsvaras av x, y respektive z? Förklara varför resultaten ser ut som de gör. Är någon konsekvent bättre med avseende på tidsåtgång?

Implementation	100k add	100k contains	5M add	5M contains
x	0.04	51	0.72	> 100 min
y	0.13	0.1	8.3	4.8
z	0.06	0.04	2.89	0.96

```
import java.util.*;
public class Timings {
   public static void main(String[ ] args) {
      // Use dierent implementations of a collection
      Collection<Integer>[ ] colls = new Collection[3];
      colls[0] = new TreeSet<Integer>();
      colls[1] = new HashSet<Integer>();
      colls[2] = new LinkedList<Integer>();
      final int NUMRUNS = Integer.parseInt(args[0]);
      final int SEED = Integer.parseInt(args[1]);
      for (Collection<Integer> c : colls) {
         Random randomizer = new Random(SEED);
         // Add a lot. . .
         long now = System.nanoTime();
         for (int i = 0; i < NUMRUNS; i++) {
            c.add(randomizer.nextInt());
         }
         System.out.println("add " + c.getClass() + "\t" + ((System.nanoTime() - now) / 1000000000.0));
         // . . .and search a lot
         now = System.nanoTime();
         for (int i = 0; i < NUMRUNS; i++) {
            c.contains(randomizer.nextInt());
         }
         System.out.println("contains " + c.getClass() + "\t" + ((System.nanoTime() - now) / 1000000000.0));
         c.clear();
      }
   }
}//Timings
```

Uppgift 5

Betrakta klasserna nedan:

```
public class Bicycle {
   public String toString() {return "Two wheeled bike"; }
}//Bicycle

public class Car {
   public String toString() {return "An ordinary car"; }
   public double getCO2EmissionLevel() { return 0.1; }
}//Car

public class RaceCar extends Car {
   public String toString() { return "Ferrarati 7.4"; }
   public double getCO2EmissionLevel() { return 5.1; }
}//RaceCar

import java.util.*;
public class Vehicles {
   public static void main(String[ ] args) {
      Collection vehicles = new HashSet();
      vehicles.add(new Bicycle());
      vehicles.add(new Car());
      vehicles.add(new RaceCar());
      for (Object obj : vehicles) {
         Car c = (Car) obj;
         System.out.println(c.toString() + " emits " + c.getCO2EmissionLevel() + " g CO2 / km");
      }
   }
}//Vehicles
```

a) När main-metoden i klassen Vehicles exekveras krashar programmet. Varför?

b) Skriv om klassen Vehicles så att felet upptäcks vid kompilering istället för vid exekveringen.

Uppgift 6

Syftet med Singleton-mönstret är att kunna garantera att *endast en instans* av en viss klass existerar. Oftast skall också denna instans kommas åt utifrån klassen. Nedan ges en klass med enbart klassmetoder. Är denna klass en singleton? Om inte, hur ska vi förändra klassen så att den blir en singeleton? Spelar det någon roll om klassen är en singelton eller inte?

```
public class GameTile {
   // code not show here
}//GameTile

public class GameUtils {
   public static GameTile[ ][ ] newBoard(final int width, final int height, final GameTile baseTile) {
      GameTile[ ][ ] board = new GameTile[width][height];
      fillBoard(board, baseTile);
      return board;
   }
   public static void fillBoard(final GameTile[ ][ ] board, final GameTile baseTile) {
      for (GameTile[ ] row : board) {
         for (int j = 0; j < row.length; j++) {
            row[j] = baseTile;
         }
      }
   }
}//GameUtils
```

Uppgift 7

Studera klassen ForwardInputIterator nedan. Som vi ser tar den bort funktionalitet snarare än lägger till.

Beskriv syftet med klassen. Om klassen fyller någon sund funktion, hade vi kunnat få samma funktionalitet om vi tänker i termer av Iterator-mönstret istället för hur Javas Iterator-gränssnitt ser ut?

Formulera lämpliga eftervillkor för de två metoderna processIterator(Iterator<T>) och processIterator(ForwardInputIterator<T>).

Vad kan gå fel vid det sista anropet till processIterator?

```
public class ForwardInputIterator<T> implements Iterator<T> {
    private final Iterator<T> it;
    public ForwardInputIterator(final Iterator<T> it) {
        this.it = it;
    }
    @Override
    public boolean hasNext() {
        return this.it.hasNext();
    }
    @Override
    public T next() {
        return this.it.next();
    }
    @Override
    public void remove() {
        throw new UnsupportedOperationException("remove() not supported.");
    }
    /**
    * @post What postcondition can this method guarantee with regards to the structure that it iterates on?
    */
    public static <T> void processIterator(Iterator<T> it) {
        // . . . do something interesting here.
    }
    /**
    * @post What postcondition can this method guarantee with regards to the structure that it iterates on?
    */
    public static <T> void processIterator(ForwardInputIterator<T> it) {
        // . . . do something interesting here.
    }
    public static void main(String[ ] args) {
        Collection<Object> coll = new LinkedList<Object>();
        // . . . add a lot of elements to coll.
        // How is this dierent . . .
        processIterator(coll.iterator());
        // From this . . .
        processIterator(new ForwardInputIterator<Object>(coll.iterator()));
        // and this . . .
        processIterator((Iterator<Object>) new ForwardInputIterator<Object>(coll.iterator()));
    }
}
```