Object oriented programming - Adam Krechowicz

1 Generic classes

Generic class do not contains information about the object type that it operates on.

```
public class Generic<T> {
  T pole;

  T getPole(){
    return pole;
  }

  void setPole(T t){
    this.pole = t;
  }

  public static void main(String args[]){
    Generic<Integer> gi = new Generic<Integer>();
    gi.setPole(1);
    System.out.println(gi.getPole());
    Generic<String> gs = new Generic<String>();
    gs.setPole("Hello");
    System.out.println(gs.getPole());
  }
}
```

2 Collections in java

2.1 Example

```
import java.util.Set;
import java.util.TreeSet;

public class ListExample {

  public static void main(String args[]){
    Set<Integer> s = new TreeSet<Integer>();
    s.add(1);
    s.add(2);
    System.out.println(s.size());
  }
}
```

2.2 Collection interface

Contains basic functionality for collections. Base interface for all objects and interfaces connected with collections.

2.3 List interface

Describes functionality to operates on lists. Implemented by ArrayList, LinkedList.

2.4 Set interface

Describes functionality for sets of objects. Contrary to lists set can not contain two equal objects. Implemented by HashSet, TreeSet.

2.5 Map interface

Described functionality for maps. Maps contains elements in form of key–value. Implemented by HashMap, TreeMap.

2.6 Types of ordering in collections

Tree collections orders elements by "greater by" relation. That is why objects stored in that collections needs to implements Comparable. Hash collections orders elements by its hashes. That is why objects stored in that collections needs to properly implements hashCode and equals methods.

```
class Element implements Comparable<Element>{
  int x;
  int y;

  public Element(int x, int y){
    this.x = x;
  }

  @Override
  public int compareTo(Element o) {
    if (this.x > o.x)
      return 1;
    else if (this.x == o.x)
      return 0;
    else
      return -1;
  }

  @Override
  public int hashCode(){
    return x;
  }

  @Override
  public boolean equals(Object o){
    if (x == ((Element)o).x && y == ((Element)o).y)
      return true;
    else
      return false;
```

```
  }

}
```

```
import java.util.Set;
import java.util.TreeSet;

public class TreeSetExample {

  public static void main(String[] args) {
    Set<Element> s = new TreeSet<Element>();
    s.add(new Element(3, 5));
    s.add(new Element(4, 5));
    s.add(new Element(6, 5));
  }

}
```

```
import java.util.HashSet;
import java.util.Set;

public class HashSetExample {

  public static void main(String[] args) {
    Set<Element> s = new HashSet<Element>();
    s.add(new Element(3,3));
    s.add(new Element(4,3));
    s.add(new Element(4,1));
  }
}
```

3 Iterators

Iterators are used to browse elements in collections. Each collection needs to implement and provide iterator that implements this interface.

```
import java.util.Set;
import java.util.TreeSet;
import java.util.Iterator;

public class ListExample {

  public static void main(String args[]){
    Set<Integer> s = new TreeSet<Integer>();
    s.add(1);
    s.add(2);
    System.out.println(s.size());

    Iterator<Integer> i = s.iterator();
    while (i.hasNext())
      System.out.println(i.next());
  }
}
```

4 Tasks to complete

1. Make familiar with documentation on above mentions classes
2. Use ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap to store primitive data types
3. Use objects of own classes to store them in collections Tree... and Hash...
4. Check how many times there are executed methods compareTo, hashCode and equals
5. Make familiar with Iterator interface documentation
6. Use iterator to browse elements in collections
7. Use iterators to remove objects from collections