Programowanie w języku Java - Adam Krechowicz

1 Klasy generyczne

Klasa generyczna nie posiada ściśle zdefiniowanego typu na jakim operuje.

```
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 Kolekcje w javie

2.1 Przykład

```
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 Interfejs Collection

Zawiera podstawową funkcjonalność kolekcji. Bazowy interfejs dla wszystkich obiektów i interfejsów związanych z kolekcjami.

2.3 Interfejs List

Zapewnia funkcjonalność do tworzenia list. Implementowany m. in. przez klasy ArrayList i LinkedList.

2.4 Interfejs Set

Zapewnia funkcjonalność dla tworzenia zbiorów. Zbór w przeciwieństwie do list nie może zawierać kilu tych samych elementów. Implementowany m. in. pzrez klasy HashSet i TreeSet.

2.5 Interfejs Map

Zapewnia funkcjonalność dla tworzenia map. Mapy przechowują dane w postaci klucz - wartość. Implementowany m. in. przez klasy HashMap i TreeMap.

2.6 Rodzaj uporządkowania w kolekcjach

Klasy Tree porządkują elementy w listach wg relacji większości. Dlatego klasy przechowywanych obiektów muszą implementować interfejs Comparable. Klasy Hash porządkują elementy wg ich haszy. Dlatego klasy przechowywanych obiektów muszą implelentować metody hashCode i equals.

```
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 Iteratory

Iteratory służą do przeglądania elementów kolekcji. Każda klasa reprezentująca kolekcję powinna zawierać własny iterator implementujący ten interfejs.

```
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 Zadania do wykonania

1. Zapoznać się z dokumentacją wyżej wymienionych interfejsów
2. Wykorzystać klasy ArrayList, LinkedList, HashSet, TreeSet, HashMap, TreeMap do przechowywania typów prostych
3. Wykorzystać własne obiekty jako elementy kolekcji Tree... i Hash...
4. Sprawdzić ile razy wowoływane są metody compareTo, hashCode i equals
5. Zapoznać się z dokumentacją intersejsu Iterator
6. Wykorzystać iteratory do przeglądania zawartości kolekcji