***Collection & Generics***

UVOD

**Java Collections Framework** podržava dva tipa kontejnera

*(****Kontejner*** *--- mjesto za pohranu nečeg):*

* ***Kolekcije*** - kontejner koji pohranjuje kolekciju elemenata.
* ***Mape*** - kontejner koji pohranjuje ključ i vrijednost koja je povezana sa ključem

***Collections*** interfejs se sastoji od metoda za rad sa kolekcijom (grupom) objekata.

Imamo:

* + - ***List*** (pristup elementima prema indexu) , ***ArrayList*** , ***LinkedList*** (ima pointer na sljedeći i prethodni čvor)
    - ***Set*** (ne može pohraniti duplikate!), ***HashSet*** (ne garantuje redoslijed), ***TreeSet*** (garantuje redoslijed), ***LinkedHashSet***
    - ***Queue*** (FIFO procesirani), ***PriorityQueue*** (prioritetno procesirani) ---- queue su kolekcije za pohranu elemenata pred procesiranje
    - ***Dequeue*** (FIFO I LIFO procesirani), ***ArrayDeque***

Ove kolekcije se razlikuju prema redoslijedu elemenata koje pohranjuju i vremena koje je potrebno da umetnu element i pristupe elementu.

SET FORSIRA HASHCODE&EQUALS CONTRACT

U setu ne možemo naći dva ista objekta.

Kako ćemo znati jesu li objekti isti?

Ako napišemo ovako:

Animal zivotinja1 = new Animal("Milka");

Animal zivotinja2 = new Animal("Milka");

System.out.println(zivotinja1.equals(zivotinja2)); //FALSE

***equals*** je metoda iz Object klase. Ona provjerava da li se dva objekta nalaze na istoj memorijskoj lokaciji.

Mi ovu metodu možemo overridati u našoj klasi, te porediti da li su dva objekta ista ako su im ista polja podataka:

*@Override*

public boolean equals(Object obj) {

if (this == obj)

return true;

if (obj == null)

return false;

if (getClass() != obj.getClass())

return false;

Animal other = (Animal) obj;

if (ime == null) {

if (other.ime != null)

return false;

}

else if (!ime.equals(other.ime))

return false;

return true;

}

Postavlja se pitanje zašto koristiti onda hashcode metodu?

Ako nam je equals dovoljna da poredimo da li su dva objekta ista?

***HashSet*** koristi haširanje da bi manipulisao podacima.

Koristi ***hashCode()*** metodu da provjeri hash vrijednosti.

***hashCode*** je metoda iz Object klase i vraća različitu int vrijednost za različite objekte.

Ako se ne override-a hashCode u klasi, a overridamo equals metodu,

po default-u poziva se hashCode metoda iz ***Object*** klase, te dva objekta iako su isti prema overridanoj equals metodi, mogu vratiti drugačiji hashCode!

ArrayListu možemo napisati ovako:

List lista = new ArrayList();

lista.add("Majmun"); //String tip podatka

lista.add(new Integer(2)); //Integer tip podatka

Što znači da možemo pohratniti objekte kojeg hoćemo tipa u jednu te istu listu.

A onda imamo sljedeći problem, hoćemo da uzmemo vrijednosti iz liste i spremimo ih u neku varijablu:

Integer broj = (Integer) lista.get(0);

//Hoću sad da pohranim onaj broj koji sam stavila u listu!

String rijec = (String) lista.get(1);

//Hoću sad da pohranim onu riječ koju sam stavila u listu!

Primijetili ste da moramo koristiti ***eksplicitnu konverziju*** zato što lista koju smo kreirali prima Object tipove.

GENERICS

Specificira tip kojeg možemo pohraniti u određenu kolekciju. Riješava gore navedeni problem..

List<String> strings = new ArrayList<>();

strings.add("string1");

String nesto = strings.get(0);

//NE MORAMO VRŠITI KONVERZIJU!

Ako imamo kod ovako napisan:

List lista = new ArrayList(); <-- ovo je raw type, tj nema specificiranog tipa.

Kako implementirati u našem kodu generics, tj narediti da se mora koristiti odgovarajući tip?

Da bismo preradili klasu lista koja implementira collection interface, moramo uraditi sljedeće:  
 public class Lista<T> implements Collections

<T> znači da moramo odrediti tip podatka kada kreiramo listu!

MAPE

Mape su strukture podataka koje nam omogućavaju brzu pretragu objekata na osnovu njihovih ključeva!

PRIMJER:

Map mapa = new HashMap();

mapa.put("ključ1" , "vrijednostKljuča1"); //Put umjesto add!

kreiranje mape koja je generics:

Map<Integer, Animal> zivotinjePremaID = new HashMap<>();

Mapa može sadržati samo objekte, ako ipak stavimo primitivnu vrijednost, onda se ona konvertuje u Objekat(auto-boxing)

***Ključ*** mora biti jedinstven(nema duplikata), a ***vrijednost*** ne mora biti jedinstvena!

***HashMap*** također koristi HashCode&&Equals contract.

Imamo:

***HashMap*** -- ako redoslijed nije važan, koristiti HashMap

***TreeMap*** -- sortira elemente po ključevima

***LinkedHashMap*** -- sortira elemente prema redoslijedu umetanja elemenata..