University of Bamberg

Java-Tutorial for ISSS-Students

Chapter 8: Collections

Chapter 8: Collections

1. Arrays
2. The Interface *java.util.Collection*
3. The Interfaces *java.util.List* and *java.util.Set*
4. Overriding methods
5. Stacks
6. Queues

Warm-up: Cars

- Create the package *chapter8* with the following classes:
  - Class *Car* with the attributes *name* (String), *price* (int), *engine* (*Engine*) + constructor, getters, setters, toString
  - Enum *Engine* with constants DIESEL_ENGINE, ELECTRIC_MOTOR and GAS_ENGINE
  - Class *Main* with main-method which creates 5 cars

Arrays

- *Arrays* contain multiple objects of the same data type – they can be created via constructor and filled with single objects one by one … :
  - *ObjectType[] arrayName = new ObjectType[numberOfObjects];*
  - *arrayName[index] = objectName;*
  - *arrayName[index] = new ObjectType(parameters);*
- … via short form that adds all objects at once, … :
  - *ObjectType[] arrayName = {Object1, Object2, … , ObjectX};*
- … or anonymously, for example as a parameter of a method:
  - *Modifiers returnValue methodName(new ObjectType[] {Object1, Object2, … , ObjectX});*

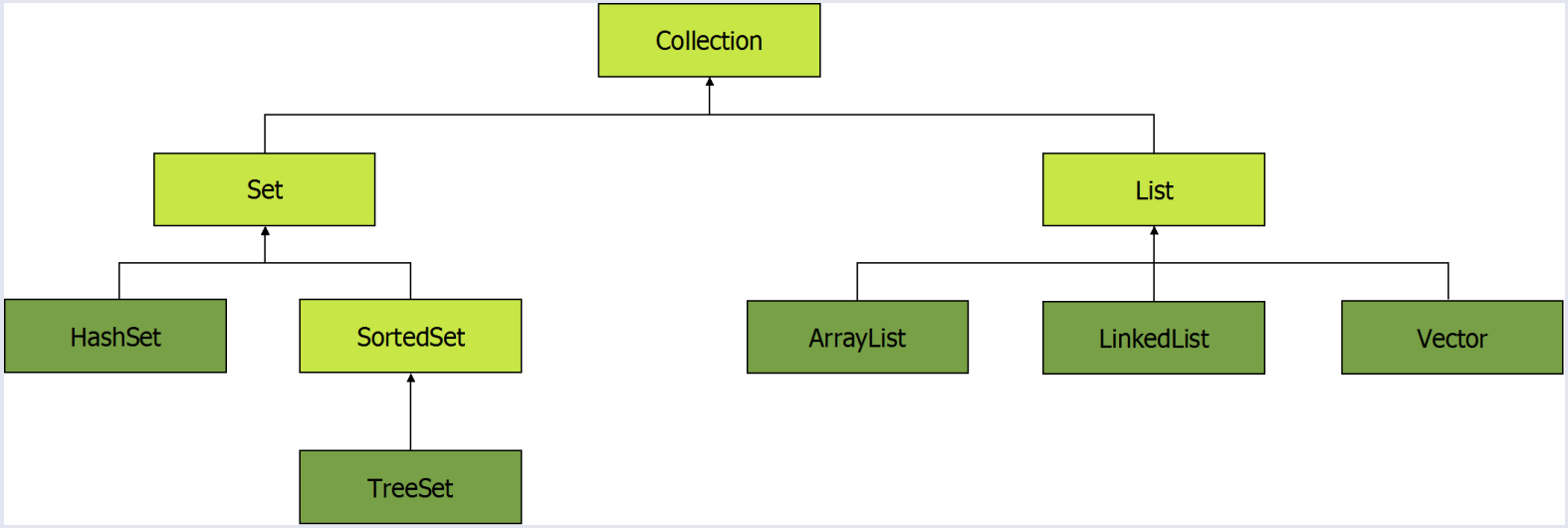
Arrays

- You can interate over Arrays via a *for* Statement … :
  - *for (int i = 0; i < array.length; i++) { code }*
- … or via the *foreach* Statement:
  - *for (ObjectType parameterName : arrayName) { code }*

Task: The car-collection

- Create the following classes:
  - Interface *CarTester* with methods *addNewCar(Car c*) which adds a new object of the class *Car* to the collection, checkCar(Car c) which returns a boolean depending on whether a given car exists in the collection or not, and *showCarCollection()* which prints every car the collection contains on the console
  - Class *CarTesterArray* which implements the Interface *CarTester* by using an Array of cars as attribute and implementing the methods of *CarTester* with operations on this Array
  - To compare two objects, use the *equals()*-method

The Interface *java.util.Collection*



The Interface *java.util.Collection*

Method	Description
add(E element)	Adds an Element of the type E to the Collection
clear()	Deletes all Elements from the Collection
contains(Object o)	Checks wether a given Object is part of the Collection
remove(Object o)	Deletes a given Object from the Collection (if it is part of it)
size()	Returns the number of Elements in the Collection

The Interfaces *java.util.List* and *java.util.Set*

- *List* is a subinterface of *Collection* that describes an ordered collection
  - *List* is implemented by classes like *ArrayList*, *LinkedList* and *Vector*
    - *List<ObjectType> listName = new ArrayList<ObjectType>();*
- *Set* is a subinterface of *Collection* that describes a collection where no duplicates are allowed
  - *Set* is implemented by classes like *HashSet* and *TreeSet*
    - *Set<ObjectType> setName = new HashSet<ObjectType>();*

Task: The car-collection, part 2

- Create the class CarTesterCollection which implements the Interface *CarTester* by using a *Collection* of cars as attribute and implementing the methods of *CarTester* with operations on this *Collection*

Overriding methods

- When working with collections, sometimes it is necessary to override some of these methods to keep your data consistent:
  - Override the *equals(ObjectType objectName)*-method so that it identifies two different objects with identical values as equal
    - Should return *true* if they are equal and *false* if they are not
  - Override the *hashCode()*-method so that it generates identical hashCodes for identical objects
    - Should return the same integer-value for identical objects
  - Override the *compareTo(ObjectType objectName)*-method to make objects comparable and therefore to be able to sort them
    - Should return -1 if the calling object is „smaller“ than the given object, 1 if it is „greater“, and 0 if both of them are identical in all the attributes that are cosidered for the comparision

Stacks (last in first out)

Method	Description
push(E element)	Adds an Element of the type E to the Stack
peek()	Returns the last inserted element of the Stack without removing it
pop()	Returns and removes the last inserted element of the Stack

Queues (first in first out)

Method	Description
add(E element), offer(E element)	Adds an Element of the type E to the Queue
element(), peek()	Returns the first inserted element of the Queue without removing it
remove(), poll()	Returns and removes the first inserted element of the Queue