**Programare Avansata pe Obiecte  
Laborator 8**

**Pahontu Bogdan-Ionut**

**E-mail:** [**pahontubogdan@gmail.com**](mailto:pahontubogdan@gmail.com)

Contents

[1. Generics 3](#_Toc6263901)

[1.1. Why use generics? 3](#_Toc6263902)

[1.2. Generic classes 3](#_Toc6263903)

[1.3. Generic naming conventions 4](#_Toc6263904)

[1.4. Generic interfaces 4](#_Toc6263905)

[1.5. Generic methods 5](#_Toc6263906)

[1.6. Bounds 5](#_Toc6263907)

[1.7. Upper-bounded wildcards 5](#_Toc6263908)

[1.8. Lower-bounded wildcards 6](#_Toc6263909)

[2. Collections 6](#_Toc6263910)

[2.1. General Overview 6](#_Toc6263911)

[2.2. Lists 7](#_Toc6263912)

[2.2.1. List implementations 7](#_Toc6263913)

[2.2.2. List methods 8](#_Toc6263914)

[2.2.3. Looping through lists 9](#_Toc6263915)

[2.3. Sets 9](#_Toc6263916)

[2.3.1. Set implementations 9](#_Toc6263917)

[2.4. Maps 10](#_Toc6263918)

[2.4.1. Map implementations 10](#_Toc6263919)

[2.4.2. Map methods 11](#_Toc6263920)

[2.4.3. Updating Map entries 11](#_Toc6263921)

[2.5. Queues 11](#_Toc6263922)

[2.5.1. Queue methods 12](#_Toc6263923)

[2.5.2. Dequeue methods (Double Ended Queue) 12](#_Toc6263924)

[2.6. Choosing the right collection type 13](#_Toc6263925)

[2.7. Comparable VS Comparator 13](#_Toc6263926)

[2.8. Cheat Sheet 14](#_Toc6263927)

[3. Tasks 14](#_Toc6263928)

# Generics

## Why use generics?

* Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.
* Elimination of casts;

// Without generics the code requires casting

List list = new ArrayList();

list.add("hello");

String s = (String) list.get(0);

// With generics

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

list.add("hello");

String s = list.get(0); // no cast

* Enable the implementation generic algorithms
  + By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

## Generic classes

* You can introduce generics into your own classes
* The syntax for introducing a generic is to declare a formal type parameter in angle brackets.

public class Crate<**T**> {

private **T** contents;

public **T** emptyCrate() {

return contents;

}

public void packCrate(**T** contents) {

this.contents = contents;

}

}

## Generic naming conventions

* Above the naming conventions:
  + **E** for an element
  + **K** for a map key
  + **V** for a map value
  + **N** for a number
  + **T** for a generic data type
  + **S** , **U** , **V** , and so forth for multiple generic types

public class SizeLimitedCrate<**T**, **U**> {

private **T** contents;

private **U** sizeLimit;

public SizeLimitedCrate(**T** contents, **U** sizeLimit) {

this.contents = contents;

this.sizeLimit = sizeLimit;

}}

* After the code compiles, the compiler replaces all references to generic types with Object.
* *Only one class file* -> there aren’t different copies for different parameterized types
* *Can`t use a primitive as a type, but you can use the wrapper (ex: Integer, Boolean)*
* *It`s not accepted to create a static variable with generics (The type is instance related)*

## Generic interfaces

* Just like a class, an interface can declare a formal type parameter:

public interface Shippable<**T**> {

void ship(**T** t);

}

* There are three ways a class can approach implementing this interface:
  + specify the generic type in the class;
  + concrete class allows the caller to specify the type of the generic
  + no use generics at all -> raw type

class ShippableRobotCrate implements Shippable<**Robot**> { // V1

public void ship(**Robot** t) { }

}

class ShippableGenericCrate<**U**> implements Shippable<**U**> { // V2

public void ship(**U** t) { }

}

class ShippableCrate implements Shippable { // V3

public void ship(**Object** t) { }

}

## Generic methods

public static <**T**> Crate<**T**> ship(**T** t) {

System.out.println("Preparing " + t);

return new Crate<**T**>();

}

## Bounds

* What we really need is a List of “whatever.” That’s what List<?> is. See the following example:

public static void printList(List<**?**> list) {

for (Object x: list) System.out.println(x);

}

public static void main(String[] args) {

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

keywords.add("java");

printList(keywords);

}

## Upper-bounded wildcards

* The **upper-bounded wildcard** says that any class that extends Number or Number itself can be used as the formal parameter type. See the following example:

public static long total(List<**? extends Number**> list) {

long count = 0;

for (Number number: list)

count += number.longValue();

return count;

}

## Lower-bounded wildcards

* The **lower-bounded wildcard** allows the usage of a specified type or a parent of that type.

public static void addSound(List<**? super String**> list) { // lower bound

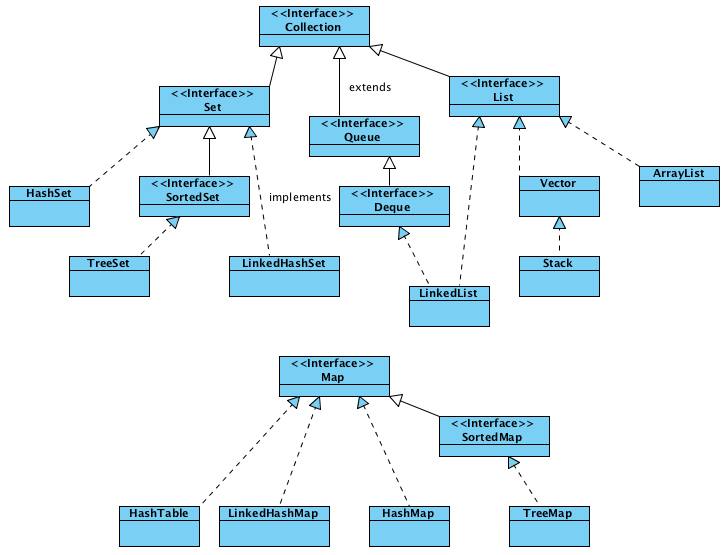
list.add("quack");

}

# Collections

## General Overview

* A collection is a group of objects contained in a single object.
* Common collections methods:
  + boolean add(E element);
  + boolean remove(Object object);
  + boolean isEmpty();
  + int size();
  + *void clear();*
  + *boolean contains(Object object);*

**

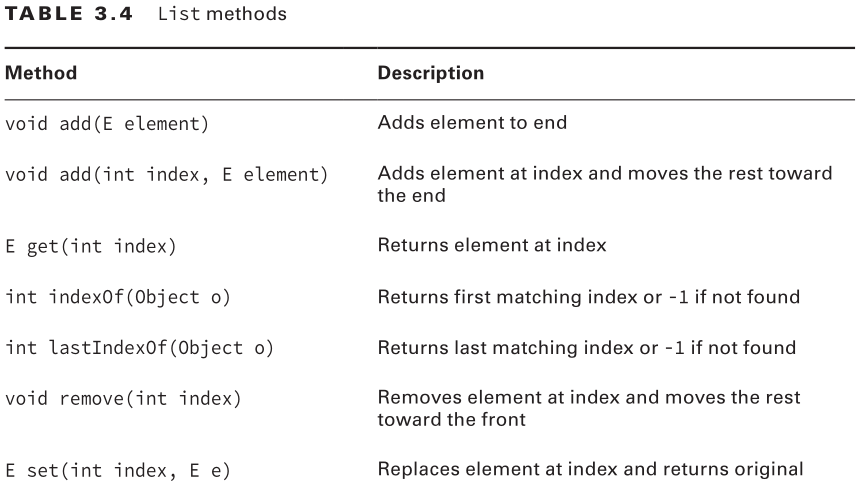
## Lists

* All List implementations are ordered and allow duplicates.
* Items can be retrieved and inserted at specific positions in the list based on an int index

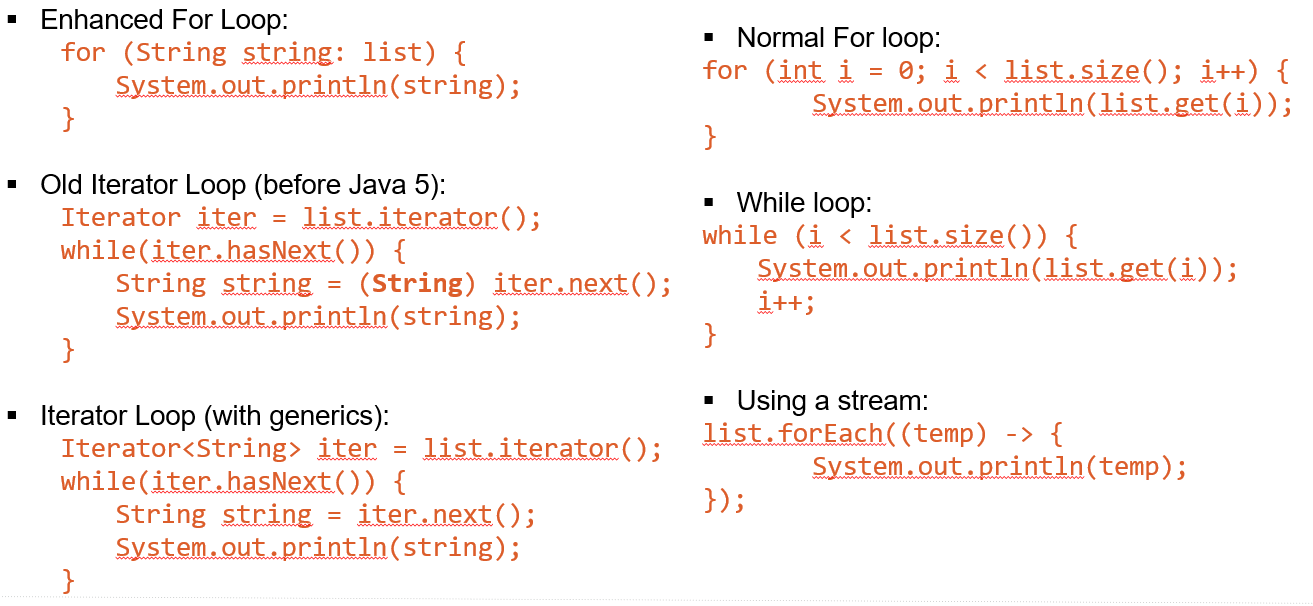
## List implementations

* + **ArrayList**
    - Like a resizable array -> the ArrayList automatically grows when elements are added.
    - The look up any element in constant time -> O(1);
    - Adding or removing an element is **slower** than accessing an element
    - Good when you are reading more often than writing to the ArrayList
    - <https://docs.oracle.com/javase/7/docs/api/java/util/ArrayList.html>
  + **LinkedList**
    - It implements both List and Queue.
    - The elements are linked using pointers and addresses. Each element is known as a node.
    - The main benefits of a LinkedList are that you can access, add, and remove from the beginning and end of the list in constant time;
    - This makes a LinkedList a good choice when you’ll be using it as Queue
    - <https://docs.oracle.com/javase/7/docs/api/java/util/LinkedList.html>

## List methods

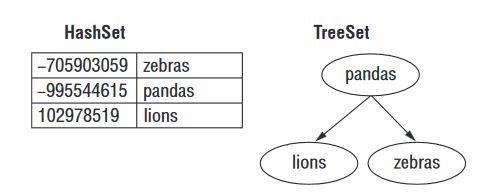


## Looping through lists



## Sets

* The ***java.util.Set*** interface is a subtype of the java.util.Collection interface.
* It represents set of objects, meaning each element can only exist once in a Set.



## Set implementations

* + **HashSet**
    - Stores its elements in a hash table.
    - It uses the hashCode() method of the objects to retrieve them more efficiently.
    - **Benefit:** adding elements and checking if an element is in the set both have constant time.
    - **Tradeoff:** you lose the order in which you inserted the elements.
    - <https://docs.oracle.com/javase/7/docs/api/java/util/HashSet.html>
  + **TreeSet**
    - Stores its elements in a sorted tree structure.
    - Implements a NavigableSet interface, which lets you slice up the collection.
    - **Benefit:** is always in sorted order;
    - **Tradeoff:** adding and checking if an element is present are both O(log n).
    - <https://docs.oracle.com/javase/7/docs/api/java/util/TreeSet.html>

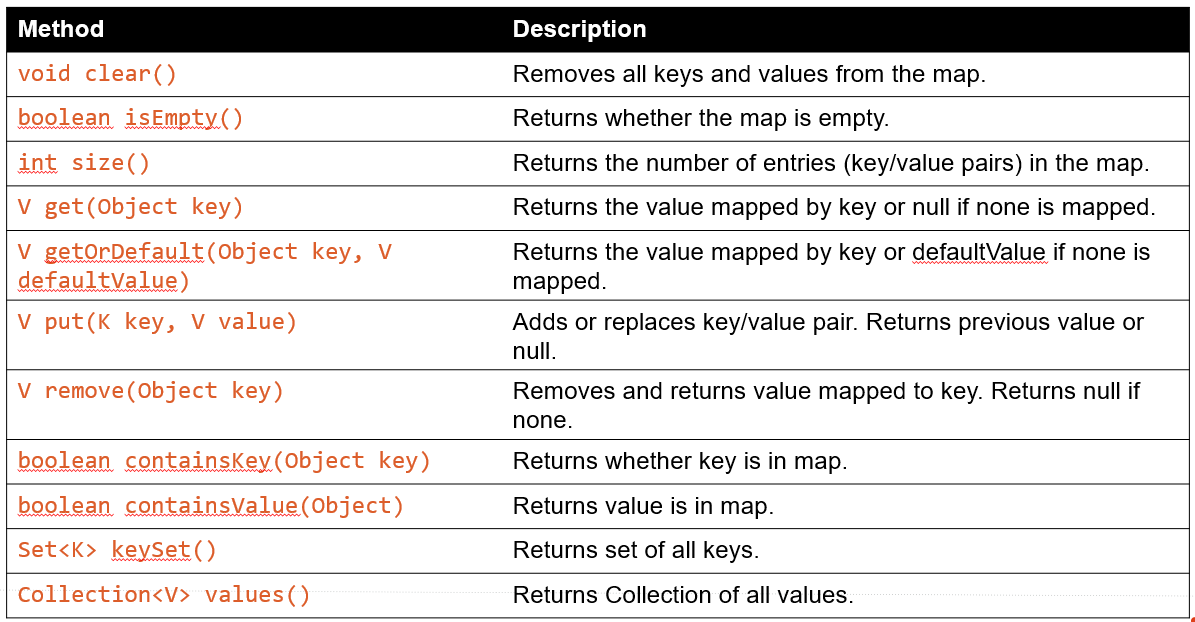
## Maps

* You use a map when you want to identify values by a key. For example, when you use the contact list in your phone, you look up “John” rather than looking through each phone number in turn.

## Map implementations

* + **HashMap**
    - Stores its elements in a hash table.
    - The main benefit is that adding elements and retrieving the element by key both have constant time. The tradeoff is that you lose the order in which you inserted the elements. Most of the time, you aren’t concerned with this in a map anyway.
    - If you were, you could use LinkedHashMap.
    - <https://docs.oracle.com/javase/7/docs/api/java/util/HashMap.html>
  + **TreeMap**
    - Stores its elements in a sorted tree structure.
    - The main benefit is that the keys are always in sorted order
    - <https://docs.oracle.com/javase/7/docs/api/java/util/TreeMap.html>

## Map methods

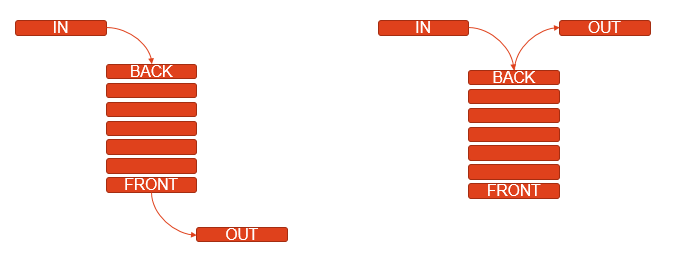


## Updating Map entries

* See updatingMapEntries.Java for a simple example

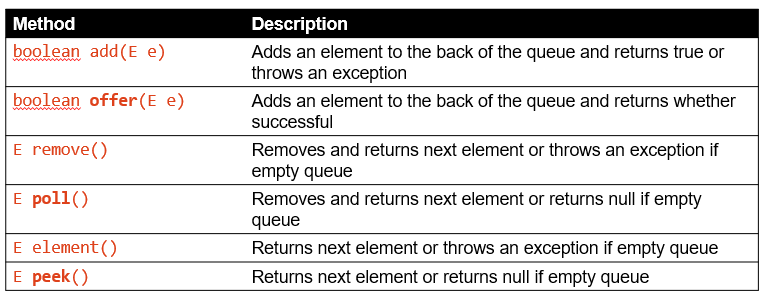
## Queues

* A Queue is a collection for holding elements prior to processing, it is used when elements are added and removed in a specific order. Unless stated otherwise, a queue is assumed to be **FIFO** (first-in, first out), another common format is **LIFO** (last-in, first-out).



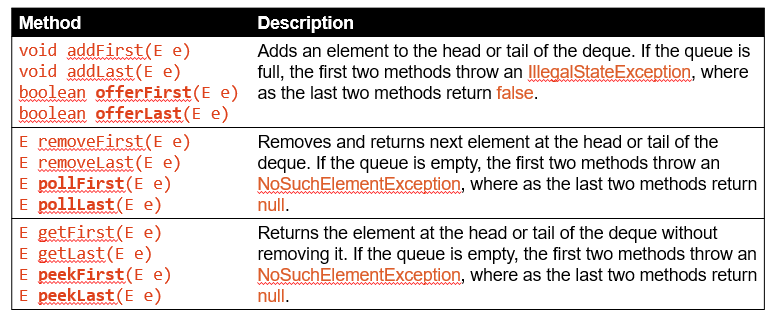
## Queue methods

* <https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html>



## Dequeue methods (Double Ended Queue)

* <https://docs.oracle.com/javase/7/docs/api/java/util/Deque.html>

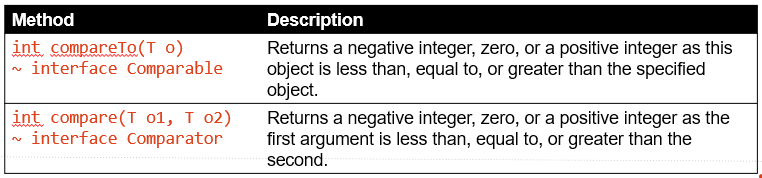


## Choosing the right collection type

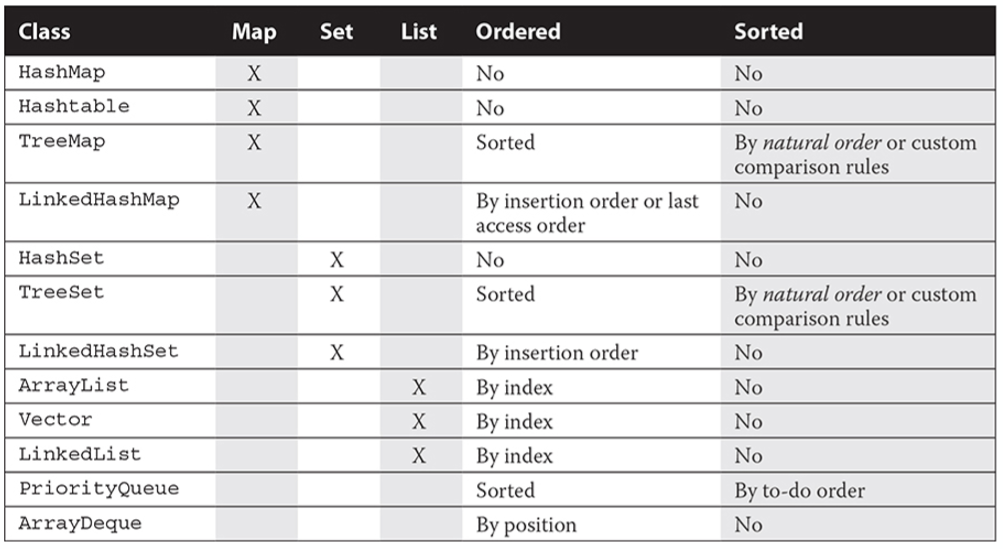
|  |  |  |
| --- | --- | --- |
| **Problem** | **Solution** | **Why** |
| To pick the top zoo map off a stack of maps | ArrayDeque | The description is of a last-in, first-out data structure, so you need a stack, which is a type of Queue. (Stack would also match this description, but it shouldn’t be used for new code.) |
| To sell tickets to people in the order in which they appear in line and tell them their position in line | LinkedList | The description is of a first-in, first-out data structure, so you need a queue. You also needed indexes, and LinkedList is the only class to match both requirements. |
| To write down the first names of all of the elephants so that you can tell them to your friend’s three-year-old every time she asks. (The elephants do not have unique first names.) | ArrayList | Since there are duplicates, you need a list rather than a set. You will be accessing the list more often than updating it, since three-year-olds ask the same question over and over, making an ArrayList better than a LinkedList. Vector and Stack aren’t used in new code. |
| To list the unique animals that you want to see at the zoo today | HashSet | The keyword in the description is *unique.* When you see “unique,” think “set.” Since there were no requirements to have a sorted order or to remember the insertion order, you use the most efficient set. |
| To list the unique animals that you want to see at the zoo today in alphabetical order | TreeSet | Since it says “unique,” you need a set. This time, you need to sort, so you cannot use a HashSet. |
| To look up animals based on a unique identifier | HashMap | Looking up by key should make you think of a map. Since you have no ordering or sorting requirements, you should use the most basic map. |

## Comparable VS Comparator

* If you want to sort your own custom objects, then the collection elements should implement Comparable interface;
* If you want to sort the list in some other way, you can use the sort method of the List interface and pass a Comparator object.



## Cheat Sheet



# Tasks

**Task 1:**

* Create a generic **Pair** class with a single generic type:
  + “first” and “second” are class members of that generic type
  + Create a default constructor and a constructor with 2 parameters for this class
  + Create getters and setters for all the class members
  + Override the toString() method for this class to display all the class members
* Create another main class that instantiates a Pair of Strings and of numbers
* Calls the Pair toString() method on both of them;

**Task 2:**

* Create a generic ArrayAlg class that gets the minimum and maximum of an array of objects of type T
  + The minMax() method -> @param an array of objects of type T
  + The minMax() method -> @return a Pair with the min and max values, or null if the input array is null or empty
  + ArrayAlg is a normal class -> NOT a generic one (only the minMax() method is generic)
* Hint: the T type needs to be comparable for this to work

**Task 3:**

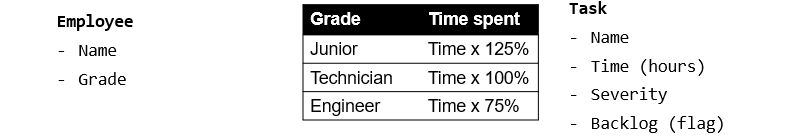
* Create a the Employee and Manager classes (Parent -> Child relationship)
* Create the FaceBook class with the printBuddies() method:
* The printBuddies() method -> @param a Pair of either Employees or Managers
* The printBuddies() method -> @return void, just print out *<Name1> and <Name2> are buddies*
* Hint: Use upper-bounded wildcards

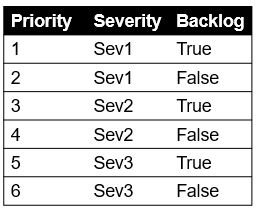
**Task 4:**

Create a program that will assign tasks to different employees for one sprint. A sprint has 10 days and in each day only 8 hours can be used for tasks.

Try to optimize the application to finish the most tasks based on priority (priority table). Note that the task may take longer or less depending on the employee grade (grade table).

After assigning the tasks, display the employees, each with their assigned tasks and total time spent on them. Create the application using queues and maps.





**Task 5:**

* Giving a string like the following:

”ala bala portocala a2t lol34 like light locos 23 55 1 af”

* The string contains separated words with space, and the words can be split into three types: alpha, numerics and alphanumerics.
* Write a method that read that type of string from a file and returns a collection with three values;
* Take into account that the input string my not respect the format so please handle the possible errors.

**Task 6:**

* Giving a text file. Extract all the words in the file, each word with its appearance frequency and print this list.