CS 250 - Fall 2021 - Lab 07: Generics

Available: Oct 13, 2021.
Due date: **Oct 18, 2021, 11:59 PM**

Generic is an important feature that is introduced from Java version 1.5. This feature allows developers to specify a parameterized type for the objects that are stored in the data structure. The names of classes or interfaces implemented with generic types are written with the syntax ClassName<E>. Inside the angle brackets, there are one or more names (separated by commas) to refer to unspecified type names.

Objectives:

The objectives of this lab are:

- Create generic methods that perform identical tasks on arguments of different types.
- Create a generic Stack class that can be used to store objects of any class or interface type.
- To understand how to overload generic methods with non generic methods or with other generic methods.
- To understand raw types and how they help achieve backward compatibility.
- To use wildcards when precise type information about a parameter is not required in the method body.

Prerequisite and Resource:

- Tutorial of Generics on Oracle Java
- Tutorial of Generics with subtype

Part 1: Print arrays of different types using non-generic

The first problem is developing methods to print arrays of integers, doubles, characters, and strings to console using non-generics and generics approach. Complete the following tasks of this part.

1. Create a new project named Lab07 and create a class named Prog1

2. Copy the following skeleton to replace the generated code in Eclipse

```
public class Prog1 {
        private static void printObjArray(Object[] arrObj) {
                //TODO add code below
        }
```

```
        private static void printArray(Integer[] arrInt) {
                //TODO add code below
        }

        private static void printArray(Double[] arrDbl) {
                //TODO add code below
        }

        private static void printArray(Character[] arrChars) {
                //TODO add code below
        }

        private static void printArray(String[] arrStr) {
                //TODO add code below
        }

        public static void main(String[] args) {
              //TODO add code below
       }
}
```

3. **Complete the method printObjArray(Object[] arrObj)**: Use a for loop that iterates through each element in the array arrObj to print such an element with a space at the end using System.out.print statement. Finally, print a newline character.

4. Add code in the **main**() method to test the method **printObjArray** with the following steps:

- Create the following arrays in main method:
  Integer [] intArr = {1, 2, 3, 4, 5 };
  Double [] dblArr = {1.1, 2.2, 3.3, 4.4};
  Character [] chrArr = {'H','E','L', 'L', 'O' };
  String [] strArr = {"once", "upon", "a", "time" };
- Now call the method printObjArray with each of the previous arrays: intArr, dblArr, chrArr, and strArr. Verify if the elements in each array are displayed on the Console.

5. Implement the four overloading printArray() methods, each one for each appropriate array type. Then, modify the code of the main() method to call the printArray() method on each of the arrays intArr, dblArr, chrArr, and strArr. Compile and run your code.

6. Now, comment out the four overloading printArray() methods to replace by a single method that uses the generic programming technique to have a single method supporting all the types and maintaining type safety. Notice the symbol E denotes the generic type that will be replaced by a specific data type when the method is called.

```
/**
 * The generic method to print an array of a generic type
```

```
 * @param arrObj: the array of the generic type E
 * @return: none
 */
private static <E> void printArray(E[] arrObj) {
        //TODO: add code below
}
```

6. Compile and rerun the program to make sure the generic method works on the arrays of different types.

Part 2: Generic search method

In Lab 6, we developed three overloading methods for linear search and binary search. These methods are required to handle different types of arrays. This section presents a generic method for searching an array of Comparable objects. The objects are instances of the Comparable interface, and they are compared using the compareTo method. We implement a program to test the method by searching an array of integers, an array of double numbers, an array of characters, and an array of strings.

Implement this program using the following steps:
1. Copy the following skeleton code to the Prog1.java file before the main() method:

```
/**
 * The generic method to search for the index of an array of a generic type
 * @param arrObj: the array of the generic type E
 * @param target: The element to search in the array
 * @return: the index of such an element in the given array, -1 if not found
 */
private static <E extends Comparable<E>> int linearSearch(E[] arrObj, E target) {
        //TODO add code below
}
```

3. **Complete the generic linearSearch method**. This method can search an element in an array of any object type, provided that the objects are also instances of the Comparable interface. The generic type is defined as <E extends Comparable<E>>. This has two meanings: a) E is a subtype of Comparable, b) it specifies that the elements to be compared are of the E type. The linearSearch method uses the compareTo method to determine the order of the objects in the array. Because the Integer, Double, Character, and String classes implement Comparable, the objects of these classes can be compared using the compareTo method.

- Use for loop with index i from 0 to less than the length of arrObj
    - If compareTo() of arrObj at the index i with target returns 0, return i
- Finally, after the for loop, return -1 to indicate the target element is not found

4. Add code to the **main()** method to call and print the result of **linearSearch** method with the following case to make sure your program works:

- Search and print the index of the integer 3 in the array intArr
- Search and print the index of the double 3.0 in the array dblArr
- Search and print the index of the character 'E' in the array chrArr
- Search and print the index of the string "a" in the array strArr

The program should display a similar output as follows:

```
1 2 3 4 5
1.1 2.2 3.3 4.4
H E L L O
once upon a time
Index of 3 in intArr: 2
Index of 3.0 in dblArr: -1
Index of E in chrArr: 1
Index of a in strArr: 2
```

Part 3: GenericStack class

In this problem, we will use the generic approach to define a generic **Stack** class that can store objects. Stack is an important data structure in computer science. The element is only inserted or removed to/from the top of the stack. The design of this data structure class is defined below:

GenericStack<E>	
- list: ArrayList<E>	An internal array list of generic data type E
+ GenericStack() + int getSize() + E peek() + void push(E obj) + E pop() + boolean isEmpty() + toString()	Constructor method to create an empty stack Return the size of the stack Return the top element in the stack (not remove) Add a new element to the top of the stack Remove and return the top element in the stack Return true if the stack is empty Return the string representation of the stack

The following steps guide you through implementing the code for this part:

1. Create a new class GenericStack in the Lab07 project

2. Copy the following code to replace the generated code of that class:

```
import java.util.ArrayList;

public class GenericStack<E> {
	//An internal array list of generic data type E
	private ArrayList<E> list = new ArrayList<>();

	/* Return the size of the stack */
	public int getSize() {
		// TODO Add code below
	}

	/* Return the top element in the stack (not remove) */
	public E peek() {
		// TODO Add code below
	}

	/* Add a new element to the top of the stack */
	public void push(E o) {
		// TODO Add code below
	}

	/* Remove and return the top element in the stack */
	public E pop() {
		// TODO Add code below
	}

	/* Return true if the stack is empty */
	public boolean isEmpty() {
		// TODO Add code below
	}

	/* Return the string representation of the stack */
	@Override
	public String toString() {
		// TODO Add code below
	}

	public static void main(String[] args) {
		// TODO Add code below
	}
}
```

3. **Complete the method getSize()**: Simply return the size of the internal array list.

4. **Complete the method peek()**: Simply return the last element of the internal array list.

5. **Complete the method push()**: Insert the element to the end of the internal array list.

6. **Complete the method pop()**:

- Get the last element of the internal array list.
- Remove the last element of the internal array list
- Return the element in step 1

7. **Complete the method isEmpty()**: Return true of the size of the internal array list is zero.

8. **Complete the method toString()**: Return "Stack: " + the string representation of the internal array list.

9. **Complete the main() method:** Add code to create two different kinds of the stack: one to store integers and another to store strings as follows:

- Create an instance named stack1 with the type GenericStack<Integer>:
  `GenericStack<Integer> stack1 = new GenericStack<>();`
- Call the push() method to put 1, 2, and 3 to stack1
- Use System.out.println to print stack1 to the Console
- Next, call the method pop() on stack1 to get the element and print it to console three times
- Next, verify that stack1 is empty now (call the isEmpty() method)
- Create another instance named stack2 with the type GenericStack<String>:
  `GenericStack<String> stack2 = new GenericStack<>();`
- Call the push() method to put "Winona", "Rochester", and "Minneapolis" to stack2
- Use System.out.println to print stack2 to the Console
- Next, call the method pop() on stack2 to get the element and print it to console three times
- Next, verify that stack2 is empty now (call the isEmpty() method)

Verify if the program should print as follows:

```
Stack: [1, 2, 3]
Pop 0 get 3
Pop 1 get 2
Pop 2 get 1
Stack1.isEmpty()=true
Stack: [Winona, Rochester, Minneapolis]
Pop 0 get Minneapolis
Pop 1 get Rochester
Pop 2 get Winona
Stack2.isEmpty()=true
```

Part 4: Find the max element of the GenericStack class

In the last problem of this lab, we develop a generic max method for finding the maximum in a stack of numbers. After that, we invoke the max method to find the maximum number in the stack in the main method.

The following steps guide you through implementing the code for this part:

1. Copy the following code to replace the generated code of that class:

```
/** Find the maximum in a stack of numbers
 * @param stack: the stack of Number (only Integer, Double, Float can be applied)
 * @return: the max number in the stack after converting to double
 */
public static double max(GenericStack<? extends Number> stack) {
        //TODO Add code below
}
```

2. **Complete the method double max(GenericStack<? extends Number> stack)**:

<? extends Number> is a wildcard type that represents Number or a subtype of Number, so it is legal to invoke max(new GenericStack<Integer>()) or max(new GenericStack<Double>()).

Use the following hints:

- Initialize a double max with the double value from the top element from the stack (Hint: call pop() on the stack then call doubleValue())
- While the stack is not empty

- double val = Get the double value of the top element from the stack
- If val is greater than max
  Set max = val
- Return max

4. **Add code to test the max method in the main() method:**

- Create another instance named stack3 with the type GenericStack<Double>:
  GenericStack<Double> stack3 = new GenericStack<>();
- Call the push() method to put 1.5, 2.5, and -3.2 to stack3
- Print the max element in stack3 by calling the max method with System.out.println
- The output should be similar as follows:
  Max element in stack 3=2.5

Lab Assignment

Complete the following assignment of this lab:

1. Similar to part 2, Implement the generic binary search on an array of generic data type E.

```
private static <E extends Comparable<E>> int binarySearch(E[] arrObj, E target) {
        //TODO add code below
}
```

2. Similar to part 4, implement the generic avg method to find the average of all numbers in a stack. Add code to the following empty method:

```
public static double avg(GenericStack<? extends Number> stack) {
        //TODO Add code below
}
```

Lab 7 Result Submission:

You need to submit the results of the following sections to the D2L assignment item of Lab07:

- Complete the required Lab Assignment

- Compress the whole Eclipse source code of your Lab07 project in zip format using the Export function