National University of Computer & Emerging Sciences (NUCES) Islamabad,

Department of Computer Science

DATA STRUCTURES – FALL 2021

LAB 02

Learning Outcomes

In this lab you are expected to learn the following:

- Generics in JAVA

Objective

To understand Generics in JAVA and its operations in a practical perspective.

Introduction to Generics:

The concept of generics in Java is somehow similar to the concept of templates in C++. Generics in Java are added to provide compile time type-safety of code and removing risk of ClassCastException at runtime which was quite frequent error in Java code. Type-safety at compile time is just a check by the compiler that the correct type is used in the correct place.

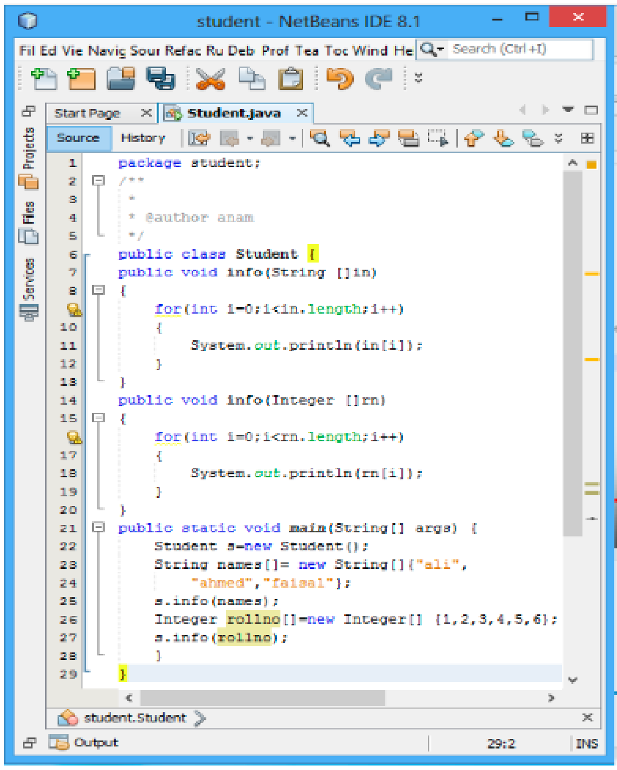
Generic Methods in Java:

Java Generic methods enable programmers to specify; a set of related methods with a single method declaration, as shown in Example 01. Please refer to this example to solve Task.

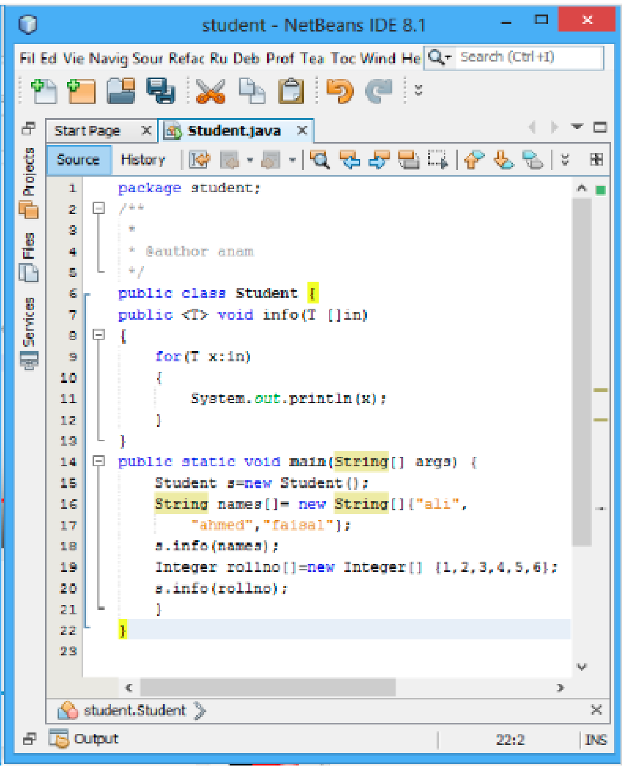
Rules:

- All generic method declarations have a type parameter section delimited by angle brackets ( ) that precedes the method's return type.
- Each type parameter section contains one or more type parameters separated by commas (<>, <>).
- Type parameters can represent only reference types (like Integer and String), not primitive types (like int, double and char).

Example 01:



```java
package student;
/**
 *
 * @author anam
 */
public class Student {
public void info(String []in)
{
    for(int i=0;i<in.length;i++)
    {
        System.out.println(in[i]);
    }
}
public void info(Integer []rn)
{
    for(int i=0;i<rn.length;i++)
    {
        System.out.println(rn[i]);
    }
}
public static void main(String[] args) {
    Student s=new Student();
    String names[]= new String[]{"ali",
        "ahmed","faisal"};
    s.info(names);
    Integer rollno[]=new Integer[] {1,2,3,4,5,6};
    s.info(rollno);
    }
}
```



```java
package student;
/**
 *
 * @author anam
 */
public class Student {
public <T> void info(T []in)
{
    for(T x:in)
    {
        System.out.println(x);
    }
}
public static void main(String[] args) {
    Student s=new Student();
    String names[]= new String[]{"ali",
        "ahmed","faisal"};
    s.info(names);
    Integer rollno[]=new Integer[] {1,2,3,4,5,6};
    s.info(rollno);
    }
}
```

Generic Classes in Java:

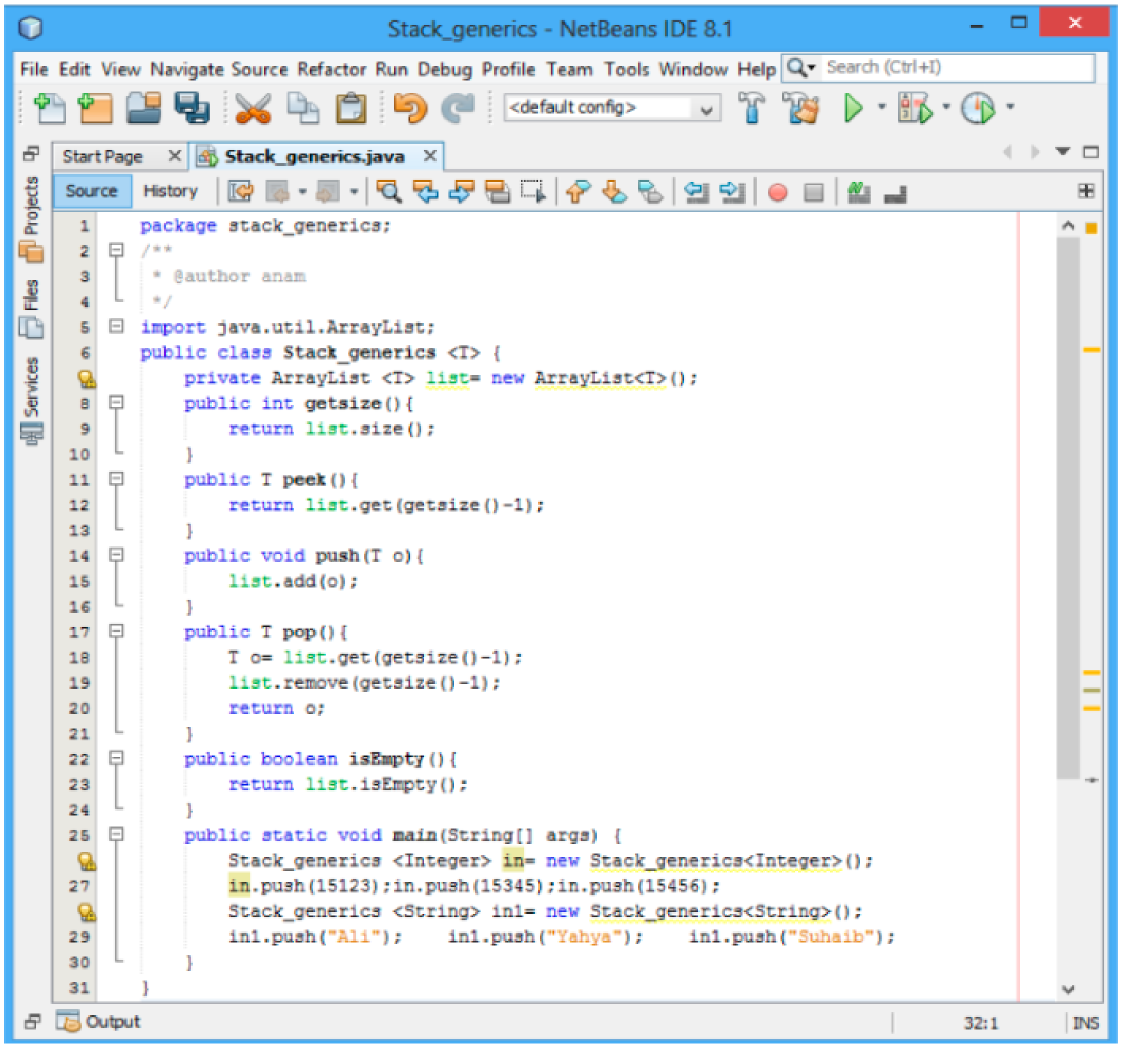
Like generic methods, Generic classes also enable programmers to specify a set of related types with a single class declaration.

Rules:

- To create a class that takes a generic, include an identifier in angle brackets immediately following the name of the class.
- Use that identifier in any variable declaration or return type in the class.
- To create an instance, it is important to parameterize the class with a concrete class to use in place of that generic.

Example 02:

In this example, the stack is implemented by using the concept of generics and collection framework. Please refer to this example for solving Task.


Stack_generics - NetBeans IDE 8.1
File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help
Search (Ctrl+I)
<default config>
Start Page
Stack_generics.java
Source
History
Projects
Files
Services
Output
32:1
INS

```java
package stack_generics;
/**
 * @author anam
 */
import java.util.ArrayList;
public class Stack_generics <T> {
    private ArrayList <T> list= new ArrayList<T>();
    public int getsize(){
        return list.size();
    }
    public T peek(){
        return list.get(getsize()-1);
    }
    public void push(T o){
        list.add(o);
    }
    public T pop(){
        T o= list.get(getsize()-1);
        list.remove(getsize()-1);
        return o;
    }
    public boolean isEmpty(){
        return list.isEmpty();
    }
    public static void main(String[] args) {
        Stack_generics <Integer> in= new Stack_generics<Integer>();
        in.push(15123);in.push(15345);in.push(15456);
        Stack_generics <String> in1= new Stack_generics<String>();
        in1.push("Ali");    in1.push("Yahya");    in1.push("Suhaib");
    }
}
```

Task 1:

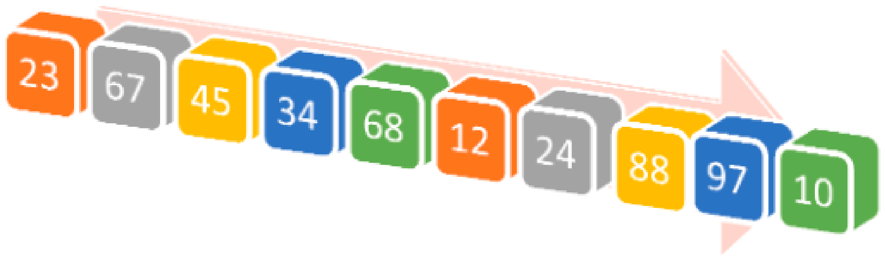
The Java **Iterator** represents an object capable of iterating through a collection of other Java objects, one object at a time. As the collection of objects being iterated can be of any data type the iterator object has to handle them all. Using the concept of Generics implement your own Iterator class. The class must contain the following methods:

- A parameterized constructor
- **hasNext()** method that checks whether the collection has more elements or not
- **next()** method that gets the next element of collection

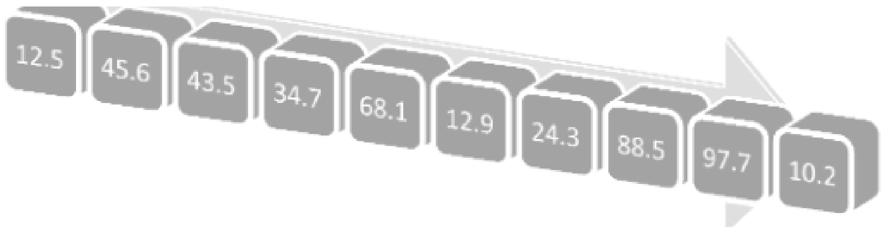
Think about the parameters and return type of these functions, considering they have to handle several data types.

Task 2:

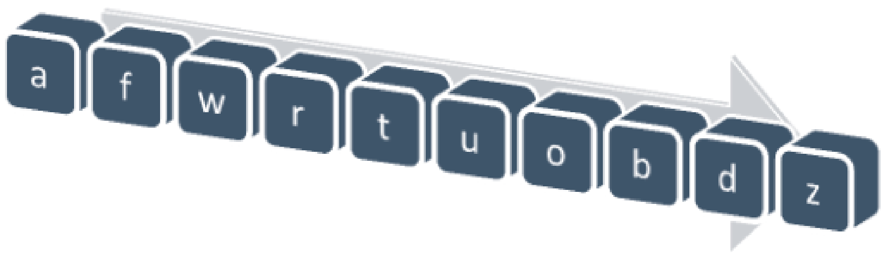
Consider a scenario of a supermarket which has received the ordered stocks; they receive the stock in the form of boxes. Each box contains some sort of label(i.e: 12, 345.67 and abc) to identify the inside material. This time they have received 30 boxes with the labels given below:



Material Type 'A'



Material Type 'B'



Material Type 'C'

Use the concept of Generics to solve the given scenario.

- Store boxes of similar label together but separate from other label types
- Display collection of each type of boxes
- Find the box with the label having the highest value.

Note: Use the iterator you have created in Task 1 for this task.

Can you compare Non-primitive Objects using regular relational operators?

You might need to extend your class with “Comparable” class to implement your comparing function.