Assignment 8

Topics: Generics

NOTE: exercises with prefix * are not mandatory**

1. Read about Generics at:
   http://docs.oracle.com/javase/tutorial/java/generics/index.html

2. *** Do the tutorial at: *http://docs.oracle.com/javase/tutorial/extra/generics/index.html*

3. Imagine we have the following classes

```
class Person {
        String name;
        Integer birthYear;

        Person(String name, int birth){
                this.name = name;
                birthYear = birth;
        }
        Integer age(){
                return birthYear;
        }
}

class Student extends Person{
        String university;
        Student(String name, int birth){
                super(name, birth);
                university = "ITU";
        }
        Student(String name, int birth, String university){
                super(name, birth);
                this.university = university;
        }
}
```

   Declare a list *lp* of *Person* objects and set it to an *ArrayList* of *Person* type.
   a. What happens if you try to insert a string into *lp*?
   b. Make statements that insert a *Person* object first and a *Student* object second into *lp*.

Declare a *Person* p and a *Student* s variable.

c. Will *lp.get(1)* return a *Person* or a *Student* object?
d. Why does *s = lp.get(1)* give a compile time error?
e. Why does *s = (Student) lp.get(1)* not give a compile time error?
f. If the right thing is to cast, what good are generics then?

Declare a list *ls* of *Student* objects. Remember, if *p* is a declared as *Person*, and *s* is declared as *Student*, one can assign as: *p = s;* and *s = (Student) p;*

g. What happens if you make the assignment *ls=lp*? Is it legal?
h. Is it legal to write *lp = ls*?

Declare a map, which can map a name to a person. Use the interface *Map* as type for the variable, and *HashMap* as the class to instantiate.

i. Make a statement that inserts a *Person* and one that inserts a *Student*.
j. Make a statement that looks up a *Person*. Can the result be assigned to *p*? Can it be assigned to *s*?

4. Define and implement a generic linked list *MyLinkedList<T>*. You do so by:

   a. adapting the class *MyIntegerLinkedList* that you created in assignment 7 for the *Integer* data type
   b. writing your own exception class that deals with exceptions thrown when there is an attempt of removing either the first element or the last element in an empty list
   c. implementing a generic *Iterator* that traverses your class *MyLinkedList<T>*; you are free to decide how traversing occurs but obviously you need to implement the methods defined in the interface *Iterator*
   d. writing a *Main* class where you test your generic implementation of a linked list. More specifically you:
      - create two linked lists: one of *Integer* elements and one of *String* elements
      - add three elements to each list and then print out their elements
      - remove the last element from the *Integer* list and then print out the list
      - remove all elements from the *String* list and then print out the list
      - remove the last element from the *String* list (notice that at this time that list should be empty!)
      - remove the first element from the *String* list (notice that at this time that list should be empty!)
      - print out the elements of the *String* list

5. Define the following class and try to figure out what it does BEFORE entering it in your IDE. After you think of the result, you should check if your guess was correct by entering the code in an IDE and, if possible, run it. Eventually, discuss the actual result.

```
public class MyGenericClass<String> extends ArrayList<String>{
   String name= "  unknown  ";

   MyGenericClass(){
```

```
        if (name.trim().toLowerCase().equalsIgnoreCase("unknown")) {
            System.out.println("Hello everybody!");
        } else {
            System.out.println("Hello " + name + "!");
        }
    }

    public static void main(String[] args) {
        new MyGenericClass();
    }

}
```

6. Define two classes, *T* and *Foo<T>*, in the same package, as the following :

```
public class T {

    public T() {
        System.out.println("Creating instance of class T (this is NOT a generics!)");
    }

    public void infoT() {
        System.out.println("I am a T!");
    }
}


public class Foo<T>{
    private T t;

    public Foo() {
        System.out.println("Creating instance of class Foo<" +
                           t.getClass().getName() + ">");
    }

    public T getT() {
        return t;
    }

    public static void main(String[] args) {
        Foo<String>foo = new Foo<>();
        T t = foo.getT();
        t.infoT();
    }

}
```

Before typing in these classes in your IDE, try to figure out the output of this program. Comment on the output (or lack thereof) of this application.
Then try to run the code above in an IDE, and, if necessary, revise your comments.

7. *** Define the class *MyGenerics<T>* as the following :

```
public class MyGenerics<T>{
   protected  T myT;

   public String getS() {
      return "Hi";
   }
   public T getT() {
      return myT;
   }
}
```

Then extend the class into *Extension* as the following class and comment the outcome (ideally, you try again to figure out the outcome before entering the code below in an IDE):

```
public class Extension extends MyGenerics {

   @Override
   public String getS() {
      return "Hi";
   }

   @Override
   public T getT() {
      return myT;
   }

   public static void main(String[] args) {
      Extension e = new Extension();
      T t = e.getT();
   }

}
```

Extend again the class *MyGenerics* and create the following class *Extension1* and comment the outcome (ideally, you try again to figure out the outcome before entering the code below in an IDE):

```
public class Extension1 extends MyGenerics<String>{
```

```
    @Override
    public String getS() {
        return "Hi";
    }

    @Override
    public String getT() {
        return myT;
    }

    public static void main(String[] args) {
        Extension1 e = new Extension1();
        T t = e.getT();
    }

}
```

Then extend one more time the class *MyGenerics* with the following class *Extension2* and comment the outcome (ideally, you try again to figure out the outcome before entering the code below in an IDE):

```
public class Extension2<String> extends MyGenerics<String>{

    @Override
    public String getS() {
        return "Hi";
    }

    @Override
    public T getT() {
        return myT;
    }

    public static void main(String[] args) {
        Extension2 e = new Extension2();
        T t = e.getT();
    }
}
```

Eventually, extend again the class *MyGenerics* with the class Extension2 as in the following and comment the outcome (ideally, you try again to figure out the outcome before entering the code below in an IDE):

```
public class Extension3<E> extends MyGenerics<String>{
```

```
    private E e;

    @Override
    public String getS() {
        return "Hi";
    }

    @Override
    public T getT() {
        return myT;
    }

    public E getE() {
        System.out.println("getE() is called!");
        return e;
    }

    public static void main(String[] args) {
        Extension3<Void> e = new Extension3();
        e.getE();
    }

}
```