Lab session #7 - Java Generic

Introduction

The goal of this lab is to explore the use of Java Generics by developing a `Pair` class that represents a 2-tuple. (Recall that a *tuple* is a finite ordered sequence of elements. A 2-tuple contains two elements, a 3-tuple contains three elements, an n-tuple contains n elements.)

A Pair class could be useful any time we need to store data that naturally occurs as an ordered pair: first and last name, two-dimensional coordinates, etc.

The challenge in designing this class is that we want to avoid specifying the element types in advance. The goal is to develop a single `Pair` class that may be used to store pairs of objects of *any* type.

Exercises

`Object` elements

One possible design involves programming the `Pair` class so that the types of the two elements are declared as `Object`. Since every reference type in Java inherits from `Object`, this approach will give us the flexibility we want.

Open the following two classes in your favorite IDE:

- ObjectPair.java
- ObjectPairTest.java

Take a few minutes to read the code, then complete the steps below.

1. Complete the `largestStadium` method so that it conforms to the Javadoc comments. Test to make sure that it works as expected. (HINT: You will need to perform some casts when you retrieve the items from the tuple.)

2. Why does this line of code compile?
   `stadiums[0] = new ObjectPair("Bridgeforth Stadium", 25000);`

   Notice that the formal and actual parameter types don't match. The expected type of the second parameter is `Object` (a reference type) and the provided argument is 1 (a primitive value). If you don't know the answer to this question Google the term "Autoboxing".

Generics

1. Create a copy of the `ObjectPair` class named `Pair`. Refactor this class to use Java generics. Your updated class should make it possible to independently specify the types of the first and second elements.
2. Create a copy of the `ObjectPairTest` class named `PairTest`. Refactor this driver so that it uses your `Pair` class. The functionality should be unchanged. The resulting code should not include any cast operations.
3. BONUS QUESTION: What happens if you re-introduce the problem that you fixed in step 1 from the previous section? Will the resulting code compile? Why do you think generic collection classes are sometimes called "type-safe" collections?
4. BONUS QUESTION: List some reasons that the `Pair` class might be preferable to the `ObjectPair` class. Can you think of any situations where the `ObjectPair` class might be preferable?

Wildcards and Subclasses

Open up the following file in a simple text editor like `vim`, `gedit` or `xed`. (***Don't open it in Eclipse!***)

CompileTest.java

Comment each assignment statement with either:

```
// C (For "Will compile.")
```

Or

```
// N (For "Will not compile.")
```

For those lines that will not compile, include an explanation of the problem. Once you have finished all of the statements, check your answers by attempting to compile the file.

<u>What to submit</u>:

Your submission should include the following:

1. Lab report to answer all above questions.
2. Source code + README (how to compile and run your code).
3. Please create a folder called "yourname_studentID_Lab7" that includes all the required files and generate a zip file called "yourname_StudentID_Lab7.zip".

Please submit your work (.zip) to Blackboard.