ECE 325 OBJECT-ORIENTED SOFWARE DES (LEC A1 Fa18)

Dashboard / My courses / ECE 325 (LEC A1 Fa18) / General / Assignment 7: Type Compatibility and Generics

Assignment 7: Type Compatibility and Generics

·Source code: *Generics*.

·Due date: **Thursday 22nd of November (5:00 pm)**. A working copy of your solution must be submitted to eClass before this date.

In this assignment, you should submit three Java files for Question 1 and2:*ArrayExample.java*,*ArrayExa* and*ArrayListExample.java*. For Question 3 the "difficult question", you should submit a folder that contains all modified files.

These questions use a hierarchy of classes of simple geometric objects:
·**GeometricShape.java**

·**TwoDShape.java**

·**ThreeDShape.java**

·**Circle.java**

·**Cone.java**

·**Rectangle.java**

·**Sphere.java**

*Test.java*is a little test class that you can run to make sure the other classes are working.

1. Java Arrays and Type Compatibility Rules

Look at the skeleton for the `main` method in*ArrayExample.java*. Add some additional statements to it so that the code compiles correctly, and when it is run, it raises a `java.lang.ArrayStoreException` when adding a cone to geoshapes (Adding a circle should be OK). Here is the exception you should get:

```
Exception in thread "main" java.lang.ArrayStoreException: Cone

at ArrayExample.main(ArrayExample.java:10)
```

Next, look at the code in*ArrayExampleGood.java*(This is exactly the same except for the classname). Add statements to this code as needed, so that the code compiles correctly, and runs without error. If you want, you can add some other statements to these `main` methods to get them printing something, but you don't need to.

2. Java Generics

Look at the skeleton code in*ArrayListExample.java*. Add additionalmethods `total_area`, `total_perime` and `add_empties`. Leave the `main` method untouched. You should get this output when you run the code:

```
Example with a list of shapes with a circle, a cone, and some empty shapes

Circle[radius=1.0]

Cone[radius=2.0, height=3.0]

Circle[radius=0.0]

Cone[radius=0.0, height=0.0]

Rectangle[width=0.0, height=0.0]

Sphere[radius=0.0]

Total number of shapes: 6


Example with a list of rectangles

Rectangle[width=2.0, height=3.0]

Rectangle[width=5.0, height=5.0]

Total number of shapes: 2

total area of rectangles: 31.0

total perimeter of rectangles: 30.0


Example with a list of 2d shapes with a circle and a rectangle

Rectangle[width=10.0, height=10.0]

Circle[radius=2.0]

Total number of shapes: 2

total area of flat shapes: 112.56637061435917


Example list of spheres
```

```
Sphere[radius=10.0]

Sphere[radius=50.0]

Sphere[radius=0.0]

Total number of shapes: 3
```

3. Difficult Question

Turn in a separate copy of all your files in a new directory (You'll need to touch many, perhaps all, of the files for the difficult question, not just*ArrayListExample.java*). Add a `supersize` method to each of your geometric shape interfaces and classes that returns a new geometric shape, of the same type as the receiver, that is twice as large in each dimension as the receiver. For example, super sizing a cone with radius 10 and height 4 should return a new cone with radius 20 and height 8. The obvious declaration for `supersize` in `GeometricShape` would be:

```
public GeometricShape supersize();
```

However, we want the type of `supersize` for `Cone` to say that it returns a `Cone`, but not a `GeometricShape`. An immediate solution is to generalize the method:

```
public interface GeometricShape {

public <T> T supersize();

}
```

Then the `supersize` method in `Cone` can be written as:

```
public Cone supersize() {

return new Cone(2.0 * radius, 2.0 * height);

}
```

However, it is not good, because we can even return a non-`GeometricShape`, which is not correct at all! So a better way is to add a type parameter:

```
public interface GeometricShape {

public voiddescribe();

public <T extends GeometricShape> T supersize();

}
```

In this condition, `supersize` can only return geometric shapes. But this is not comprehensive, either. For example, it is possible that a `Rectangle.supersize` returns a circle.

Make the appropriate modifications to all the geometric shape classes and add amethod `supersize_lis` so that it takes an array list of some kind of geometric shapes (for example, rectangles) and returns an array list of the same type, with the shapes super sized. The skeleton code in*ArrayListExample.java*includes some commented-out lines at the end that use the `supersize_list` method. Uncomment them (but don't otherwise change them). You should get the following additional output:

```
super-sizing a list of rectangles

Rectangle[width=4.0, height=6.0]

Rectangle[width=10.0, height=10.0]

Total number of shapes: 2


super-sizing a list of spheres

Sphere[radius=20.0]

Sphere[radius=100.0]

Sphere[radius=0.0]

Total number of shapes: 3
```

Hint: starts with generalizing the `GeometricShape` :

```
public interface GeometricShape<T extends GeometricShape<T>> {

public voiddescribe();

public T supersize();

}
```

Generics.zip

Submission status

Attempt number	This is attempt 1 (1 attempts allowed).
Submission status	Submitted for grading
Grading status	Graded
Due date	Thursday, 22 November 2018, 5:00 PM
Time remaining	Assignment was submitted 4 hours 28 mins early
Last modified	Thursday, 22 November 2018, 12:31 PM

File submissions	ArunWoosaree.zip ✚ Export to portfolio
Submission comments	✚ Comments (0)

Edit submission

Make changes to your submission

Feedback

Grade	30.00 / 30.00
Graded on	Wednesday, 28 November 2018, 5:50 PM
Graded by	Mona Nashaat Ali Elmowafy