SWEN20003
Object Oriented Software Development
Workshop 7

Rohyl Joshi

Semester 1, 2020

Discussion

UML

1. What is a UML class diagram and why are they useful?
2. How is privacy represented in UML class diagrams?
3. How do we label types in UML?
4. How do we represent the following concepts in UML? What do they mean?
    - Association
    - Generalisation
    - Realisation
    - Multiplicity
    - Abstract

Generics

1. What are the advantages of using generically typed classes and methods?
2. What types can be used for generic type parameters?
3. Why do we even have generics? Why can't we just make our methods/constructors take in `Object` and use explicit typecasting to handle all the types?

Design

UML

- Create a UML diagram to represent Workshop 5's assessable problem.
- The game of Monopoly is defined by a *board*, which contains 40 *spaces*, and between 2 to 6 *players.*

  A space can be either a *property*, *chance*, or *bonus*, and each of the types has a different *action* when a player lands on them. Properties may additionally be *railway stations* or *utilities.*

  Players each have a position on the board, an amount of money that they have, a number of properties that they own, and can move along the board.

Generics

- Create a class `Box` that accepts two type parameters `T` and `U`, and simply stores two objects (one of type `T` and one of type `U`). The class should take in the two objects in its constructor, and expose access to the objects through getters.
- The `Number` abstract class is part of the Java standard library and is the superclass of all numeric primitive wrapper classes such as `Integer`, `Double`, `Long`, e.t.c. Using this knowledge, create a new class `BoundingBox` that builds on top of your `Box` class design but only allows subclasses of `Number` to be stored in the box.
  Hint: use a bounded type parameter.

Implementation

Begin Project 2. Be sure to read the project description carefully, and clarify any issues with your tutor. Remember that the first submission for the project is your *design* document; we don't expect you to start programming until after this submission (although you can if you're confident in your design).

Assessable Question

A *tree* is a data structure that is made up of **nodes**. At the most basic level, each node holds a reference to its children, and a value. A *binary tree* is a tree that has at most two children, conventionally referred to as 'left' and 'right'.

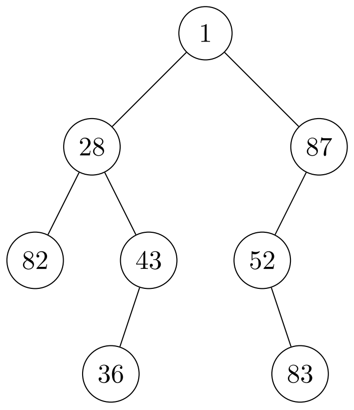

Figure 1: A simple binary tree

Figure 1 shows an example of a binary tree storing integer values. The class `BinaryTreeNode` provides you with an implementation of a binary tree node that can store an integer. Trees are commonly represented using integers as the value type, but in practicality we can store any data type!

Your Task

Modify the implementation of `BinaryTreeNode` so that a binary tree node can hold **any data type** instead of just integers. You must use generics to achieve this task. You must not add attributes, change the names of methods, add methods, remove methods, or change the body of methods. You should not be using any libraries apart from what is provided.

Package

The assessable problem package provides you with the `BinaryTreeNode` class that can only store integers, and a testing class `BinaryTreeNodeTest`. **The testing class will initially be filled with type errors**, that is expected, do not try to resolve them directly. When you have finished making `BinaryTreeNode` generic, the errors should disappear and you should be able to run your tests. Do not modify the testing class.

Submission

You will submit your solution to your workshops repository. At the very least, you should maintain the following structure:

```
username-workshops
└─ workshop-7
    └─ src
        └─ BinaryTreeNode.java
```

As always, you can use the example repository here as a reference. This assessable question is due on **Saturday the 23rd of May at 11:59pm**. This has been pushed back so it does not conflict with your Project 2A due date.

This is a rather straightforward problem testing a very basic understanding of generics, it shouldn't take too long if you've learned the content.