CMSC131

Intro to Generic Concept
Example Data Structure: Pair

A Simple Problem
What if I asked you to write a program that had various methods that needed to return a pair of things of the same data type?
General Purpose Data Structures

Imagine building a `Pair` class with the ability to store references to two objects of the same data type.

The constructor could get references to the two objects. The getters could return references to them. A single object could “hold” two things.

– However, we’ve seen that Java needs everything we declare to have a specific type which can be used at compile time for checking for valid commands and used at execution time to allocate the correct amount of space in memory.

Polymorphism and Generics

One of the powerful paradigms used in object oriented programming is being able to have a common structure that can be used with a wide variety of data types. This is one form of polymorphism.

In Java, this can be supported by something called `Generics` to effectively build a template than can be instantiated in many ways.
**Pair<Type>**

**Pair** is an example class that I’ve created for demonstration purposes, though it could be useful. It can hold two references to any type of object you specify.

- It is a **Generic** structure, and you can specify what `<Type>` of object references a specific instance of **Pair** will be allowed to hold.
- This can be better than a data structure designed to simply allow references to *any* object to be stored since this approach will allow for compile-time type checking and make it easier to invoke methods using the objects that were stored in the list.

**Pair behaviors/properties**

When created, a **Pair** object’s constructor must be sent two references (to objects of the proper type).

Once a **Pair** object exists, we have getters `getRefToFirst()` and `getRefToSecond()` that can be invoked by the **Pair** object.

We now have a way to return a pair of object references as the return value of a method even though Java only supports returning one value – we return a reference to a single object that contains references to two others!
Generics and Primitives

Recall that every primitive type (int, double, etc.) as an object-based wrapper version in Java (Integer, Double, etc) and that Java will do a variety of automatic boxing and unboxing of these.

We could make use of this aspect of Java by (for example) being able to create a Pair of Double.

Declaring and creating a Pair

//Declare a reference.
Pair<Student> twoStudents;

//Create some Student objects.
Student s1 = new Student("Pat");
Student s2 = new Student("Sam");

//Create a new Pair object to hold // references to them and store the // reference to that new object.
twoStudents = new Pair<Student>(s1,s2);
Using the **Pair**

Recall, the **Pair** has instance methods that allow you to access the items.

```
System.out.println("First was " +
twoStudents.getRefToFirst());

System.out.println("Second was " +
twoStudents.getRefToSecond());
```

Note, in my implementation the constructor and getters just copy the references. These are what we have called shallow copies.

---

Copyright © 2016-2017: Evan Golub