Java Review: ArrayList Object Types Explained

CS 2100

Adapted from: https://www.learnhowtoprogram.com/java/ Data Structures: ArrayList

Determine what *type* of data the ArrayList will contain.

Data in Java ArrayLists isn't *required* to be one single type like it is in an `Array`. However, if your ArrayList's data *will* be a single type, it is best practice to declare that type in angle brackets, like so:

```
import java.util.ArrayList;


ArrayList<String> myStringList = new ArrayList<String>();

ArrayList <Integer> myIntegerList = new ArrayList<Integer>();

ArrayList <Boolean> myBooleanList = new ArrayList<Boolean>();
```

Generics

The `String`, `Integer` and `Boolean` keywords we see in the angle brackets `<>` above are known as generics. **Generics** were introduced in 2004 as part of JDK 5.0 in order to allow the Java compiler to catch and alert us of more errors while compiling code, *before* we ever run our programs. Generic types tell Java what *kind* of object this data structure can hold.

The generic type can be any object type. **Object types** are simply data types that *are objects*. They're easy to spot because their first letter is capitalized. Since primitives are *not* objects, you'll have to use their object type wrapper class when creating an ArrayList instead (ie: use `Integer` instead of `int`. `Integer` is an object type, and `int` is a primitive).

It is generally a good idea to use the generic type for a couple reasons:

- Preventing your code from breaking: You may want to iterate through your list and call String specific methods on each item. If you had an Integer stored in the list your application would break.
- It allows for better code clarity, so other programmers reading or using your code know what they are allowed to do with each list.
- They allow the compiler to catch more errors during compilation, *before* you run your program. This saves you time tracking down pesky bugs.

You will find that the following information might not make sense right away. As we learn more about Java (especially the Object class and Inheritance, … etc.) over the following weeks, you can come back to this and hopefully it will make more sense. (In other words, if it doesn't make sense now, *don't worry!*)

Storing Varying Data in ArrayLists

Earlier we mentioned "data in Java ArrayLists isn't *required* to be one single type" (although often it is.) This section will briefly explain how to store a variety of *differing* data types in a single `ArrayList`.

To place objects of *different types* in the same `ArrayList`, declare the ArrayList's generic as `<Object>`. That is, declare the data type to be "Object" within the angle brackets, like so:

```
ArrayList<Object> objectList = new ArrayList<Object>();
```

All objects inherit from a built-in Java class called `Object`. This is why we can simply declare their type as `Object`, because they *are* instances of the `Object` class, too. We'll learn more about the concept of inheritance later, just know any object can use methods defined by the `Object` class, or be stored in an `Object` type variable.

A Word of Warning Regarding `Object` Types

However, if you declare `Object` as the generic type **you will no longer be able to use class-specific methods on the contents of the array list**. By declaring everything in the `ArrayList` as the more general `Object`, you lose the ability to call methods meant for more specific classes like `String`.

For example, a string saved as an `Object` in an `ArrayList`, like this....

```
ArrayList<Object> objectList = new ArrayList<Object>();

objectList.add("Hello");
```

...Can no longer have `String`-specific methods like `.toUpperCase()` called on it. You can still use methods defined by the `Object` class, though, like `toString()`, `.equals()` and `.getClass()`.