12.5 Chapter Summary

Summary of the main points in this chapter.

You can find the code from this lecture here.

Exceptions

Most likely you have encountered an exception in your code such as a `NullPointerException` or an `IndexOutOfBoundsException`. Now we will learn about how we can “throw” exceptions ourselves. Here is an example of an exception that we throw:

```
throw new RuntimeException("For no reason.");
```

This is useful to ensure reasonable functioning of our code, even when facing unexpected behavior.

Iteration

Difference between Iterators and Iterables

These two words are very closely related, but have two different meanings that are often easy to confuse. The first thing to know is that these are both Java interfaces, with different methods that need to be implemented. Here is a simplified interface for Iterator:

```
public interface Iterator<T> {
  boolean hasNext();
  T next();
}
```

Here is a simplified interface for Iterable:

```
public interface Iterable<T> {
    Iterator<T> iterator();
}
```

Notice that in order for an object (for example an ArrayList or LinkedList) to be *iterable*, it must include a method that returns an *iterator*. The iterator is the object that actively steps through an iterable object. Keep this relationship and distinction in mind as you work with these two interfaces.

Object Methods

toString

The `toString()` method returns a string representation of objects. For example, `System.out.println(someObject)` calls the `toString()` method of `someObject`, and prints to console whatever string it returns.

This is most helpful when we are debugging, as it allows us to much more easily understand the current state of our Objects.

== vs .equals

We have two concepts of equality in Java- "==" and the ".equals()" method. The key difference is that when using ==, we are checking if two objects have the same address in memory (that they point to the same instance or object). On the other hand, .equals() is a method that can be overridden by a class and can be used to define some custom way of determining equality. This permits the class to utilize the additional knowledge it has about itself to more accurately answer questions of equality.

For example, say we wanted to check if two stones are equal:

```
public class Stone{
  int weight;
  public Stone(int weight){
    this.weight = weight;
  }
}
Stone s = new Stone(100);
Stone r = new Stone(100);
```

If we want to consider s and r equal because they have the same weight. If we do check equality using ==, these Stones would not be considered equal because they do not have the same memory address.

On the other hand, if you override the equals method of Stone as follows

```
public boolean equals(Object o){
  return this.weight == ((Stone) o).weight
}
```

We would have that the stones would be considered equal because they have the same weight.

Previous
12.4 Object Methods

Next
12.6 Exercises