Quiz Preparation for Week 8

Concepts

- Abstract
- Interface
- Anonymous Class
- Lambda Expression

EMP Session Links

1. EMP 10-15: Topics including reference/abstract/interface

Quick Concept Overview & Practice

Abstract

1. Can be applied to both classes and methods, e.g.:

```
public abstract class Shape {
  // remember, abstract method doesn't have a body
  public abstract int sides();
}
```

2. An *abstract class* may or may not contain *abstract methods*, but if you want to have *abstract method(s)*, the class needs to be abstract. The following is wrong:

```
public class Shape {
  public abstract int sides();
}
```

3. An *abstract class* cannot be instantiated (you can't create an instance for an abstract class).
4. An *abstract class* can be extended (you can have another class extends an abstract class and calls its methods). Children classes need to implement the abstract methods, else they also need to be abstract.

Practice

Interface

1. A java class can only extends from one parent, but it can implement multiple interfaces. It means it supports the functionalities declared in the interfaces. E.g.:

```
interface A {
  int funcA();
}
interface B {
  int funcB();
}
public class Parent { }
public class Alphabet extends Parent implements A, B {
```

```
  public int funcA() {
    return 1;
  }
  public int funcB() {
    return 2;
  }
}
```

2. Interface works similarly with polymorphism/inheritance. You can have a variable of an interface type and it can refer to the classes that implements the interface. E.g.:

```
// given the codes in (1)
A a = new Alphabet();
// this code returns 1
int some = a.funcA();
// this code gives error because interface A does not have funcB() supported
int other = a.funcB();
```

3. Comparable is a built-in java interface that allows you to implement the *compareTo()* method for instance comparison.
4. Iterator, Iterable are two other built-in java interfaces that allow you to implement methods for enhanced for-loop (iterating items).

Anonymous Class

1. An *anonymous class* does not have a name, must extend a class or implement an interface, is created immediately, and cannot provide a constructor. Syntax for example:

```
interface Test {
  void test();
}
Test t = new Test() {
  @Override
  public void test() { }
};  // don't forget the semi-colon!
t.test();
```

2. *Anonymous classes* can be used to flexibly create inline instances that perform certain tasks the caller defines. Please review the lecture on 10/15/2020 - "Uses for Anonymous Classes".

Lambda Expression

1. Functional programming is a style of programming for which programs are constructed by applying and composing functions, while object-oriented programming (e.g. Java) is a style for which programs are constructed by manipulating objects.
2. Java cannot store functions in variables (as opposing to what functional programming). However, Java can achieve some similar expression style called lambda expression. For example:

```
interface Increment {
  int increment(int value);
}
// method 1 - using anonymous class
Increment a = new Increment() {
  @Override
  public int increment(int value) {
    return value + 1;
  }
```

```
};
// method 2 - using lambda expression
Increment b = (value) -> value + 1;

// the following codes both print 11
System.out.println(a.increment(10));
System.out.println(b.increment(10));
```

Combining it all together!