CSC 243 - Java Programming

Java 8 Features

Functional Interfaces

- A functional interface, also called a single abstract method (SAM) interface, has only one abstract method declared in the interface definition
- An interface can declare abstract methods from `java.lang.Object` and still be considered a functional interface
- The `@FunctionalInterface` annotation can be used to enable compile time checking of the interface
- The `java.util.Function` package contains many functional interfaces
- Example:

```
@FunctionalInterface
public interface ExampleFunctionalInterface {
    public void doTheThing();
    public String toString();
}
```

Default Methods

- The `default` keyword can be used to provide a default implementation of an interface method
- When a class implements multiple interfaces with the same default methods the compiler cannot resolve which method to call. There are two ways to handle the ambiguity:
  - The class overrides the default implementation
  - The class calls the default method of the specified interface using the `super` keyword

Default Method Example

```
public interface One {
    default void print() {
        System.out.println("One");
    }
}

public interface Two {
    default void print() {
        System.out.println("Two");
    }
}

public class C implements One, Two {
    default void print() {
        Two.super.print();
    }
}
```

Lambda Expressions

- Lambda expressions are typically used to define inline implementations of functional interfaces
- Basic syntax:
  `(parameter, [parameters]) -> {expression body}`
- Additional characteristics:
  - Type declarations are optional
  - The parenthesis are optional if there is only one parameter
  - The curly braces around the body are optional if body contains a single statement/expression
  - The `return` keyword is optional if the body contains a single expression

Method References

- A method reference is shorthand syntax for a lambda expression that executes one method. The general syntax is: `Object::methodName`
- A method reference can be used for the following types of methods:
  - Static method:
    `(args) -> Class.staticMethod(args)`
    `Class::staticMethod`
  - Instance method of an object of a particular type:
    `(obj, args) -> obj.instanceMethod(args)`
    `ObjectType::instanceMethod`
  - Instance method of an object of an existing type:
    `(args) -> obj.instanceMethod(args)`
    `obj::instanceMethod`
  - Constructor:
    `(args) -> new ClassName(args)`
    `ClassName::new`

Streams

A stream represents a sequence of objects from a source with the following characteristics:

- A stream provides a sequence of elements where the elements are generated on demand
- The source of a stream can be a Collection, Array, or I/O resource
- The stream supports aggregate operations
- Most stream operations return the stream itself allowing stream operations to be pipelined
- Stream operations provide implicit iteration