Section 5 - Interfaces and Lambda Expressions
Quiz Answers

1.

Methods in interfaces can have no implementations whatsoever, whereas abstract classes let you mix abstract methods and non-abstract methods. You can implement multiple interfaces, but you can only subclass one class. Interfaces, then, should be used when you're representing behavior that can be applied to many different kinds of classes (a move() method in an interface, for example, could apply to people, animals, or machines), while abstract classes should be used when you need subclasses to provide implementations (an operate() method in an abstract base class called Robot, for example, should be implemented in a DiggerRobot or DrillerRobot, but it doesn't make sense to put this operate method in an interface because it only applies to robots, not people or animals).

2.

Anonymous inner classes are useful because they let you create a subclass of an abstract class or interface without having to actually create a new class. This makes your code cleaner and easier to read.

3.

If you define an interface, then classes that implement it need to provide implementations for every method in it. If your class is used throughout your program many times, then if you add a new method to the interface, every implementer will have to add that method. By adding the default keyword, you can provide a default implementation for the new method instead so that implementers don't have to provide an implementation for the new method.

4.

Lambda expressions are just wrappers around anonymous inner classes (you can prove this to yourself by replacing a lambda expression with an anonymous inner class; your program will compile just fine). They make your code cleaner and easier to read.