Inner Classes

- A "regular" inner class is declared inside the curly braces of another class, but outside any method or other code block.
- An inner class is a full-fledged member of the enclosing (outer) class, so it can be marked with an access modifier as well as the abstract or final modifiers (but of course, never both abstract and final together— remember that abstract means it must be subclassed, whereas final means it cannot be subclassed).
- An inner class instance shares a special relationship with an instance of the enclosing class. This relationship gives the inner class access to all of the outer class' members, including those marked private.
- To instantiate an inner class, you must have a reference to an instance of the outer class.
- From code within the enclosing class, you can instantiate the inner class using only the name of the inner class, as follows:

  MyInner mi = new MyInner();

- From code outside the enclosing class' instance methods, you can instantiate the inner class only by using both the inner and outer class names, and a reference to the outer class as follows:

  MyOuter mo = new MyOuter();

  MyOuter.MyInner inner = mo.new MyInner();

- From code within the inner class, the keyword this holds a reference to the inner class instance. To reference the outer this (in other words, the instance of the outer class that this inner instance is tied to) precede the keyword this with the outer class name as follows:

  MyOuter.this;

Method-Local Inner Classes

- A method-local inner class is defined within a method of the enclosing class.For the inner class to be used, you must instantiate it, and that instantiation must happen within the same method, but after the class definition code.
- A method-local inner class cannot use variables declared within the method (including parameters) unless those variables are marked final.
- The only modifiers you can apply to a method-local inner class are abstract and final. (Never both at the same time, though.)

Anonymous Inner Classes

- Anonymous inner classes have no name, and their type must be either a subclass of the named type or an implementer of the named interface.
- An anonymous inner class is always created as part of a statement, so don't forget to close the statement after the class definition, with a curly brace. This is one of the rare times you'll see a curly brace followed by a semicolon in Java.
- Because of polymorphism, the only methods you can call on an anonymous inner class reference are those defined in the reference variable class (or interface), even though the anonymous class is really a subclass or implementer of the reference variable type.
- An anonymous inner class can extend one subclass, *or* implement one interface. Unlike non-anonymous classes (inner or otherwise), an anonymous inner class cannot do both. In other words, it cannot both extend a class *and* implement an interface, nor can it implement more than one interface.
- An argument-local inner class is declared, defined, and automatically instantiated as part of a method invocation. The key to remember is that the class is being defined within a method argument, so the syntax will end the class definition with a curly brace,

followed by a closing parenthesis to end the method call, followed by a semicolon to end the statement:

```
});
```

Static Nested Classes

- Static nested classes are inner classes marked with the static modifier.
- Technically, a static nested class is *not* an inner class, but instead is considered a top-level nested class.
- Because the nested class is static, it does not share any special relationship with an instance of the outer class. In fact, you don’t need an instance of the outer class to instantiate a static nested class.
- Instantiating a static nested class requires using both the outer and nested class names as follows:

  BigOuter.Nested n = new BigOuter.Nested();
- A static nested class cannot access nonstatic members of the outer class, since it does not have an implicit reference to any outer instance (in other words, the nested class instance does not get an *outer this* reference).