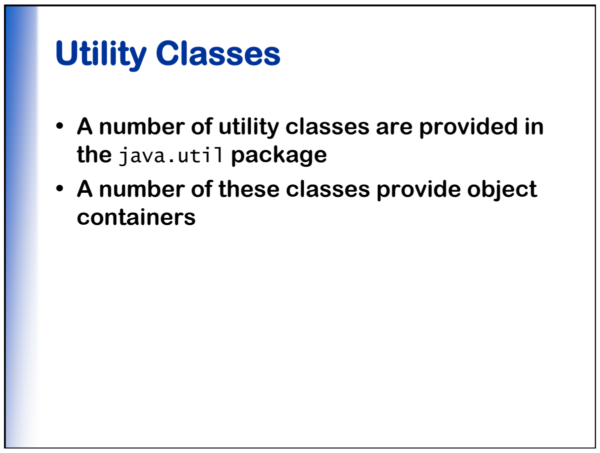
Week 4

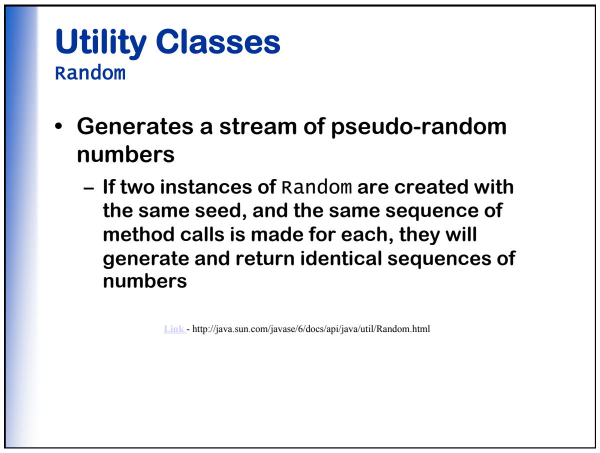
Assignment Three

Additional Libraries



Utility Classes

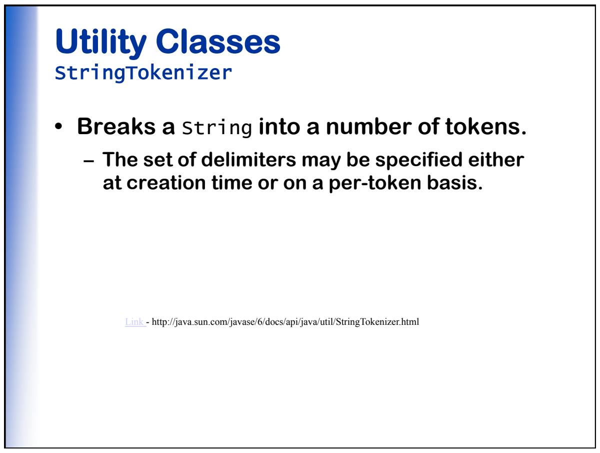
- A number of utility classes are provided in the `java.util` package
- A number of these classes provide object containers



Utility Classes
`Random`

- Generates a stream of pseudo-random numbers
  - If two instances of `Random` are created with the same seed, and the same sequence of method calls is made for each, they will generate and return identical sequences of numbers

Link - http://java.sun.com/javase/6/docs/api/java/util/Random.html



Utility Classes
`StringTokenizer`

- Breaks a `String` into a number of tokens.
  - The set of delimiters may be specified either at creation time or on a per-token basis.

Link - http://java.sun.com/javase/6/docs/api/java/util/StringTokenizer.html

Utility Classes
Collections

- A collection represents a group of objects, known as its elements.
  - Duplicate elements may or may not be allowed.
  - A collection may be ordered or unordered.
  - Classes and interfaces are provided for different types of collections and traversing the items in the list.

Utility Classes
`Iterator<E>`

- Interface for traversing collections
  - Allows for the removal of an object from the collection during the iteration

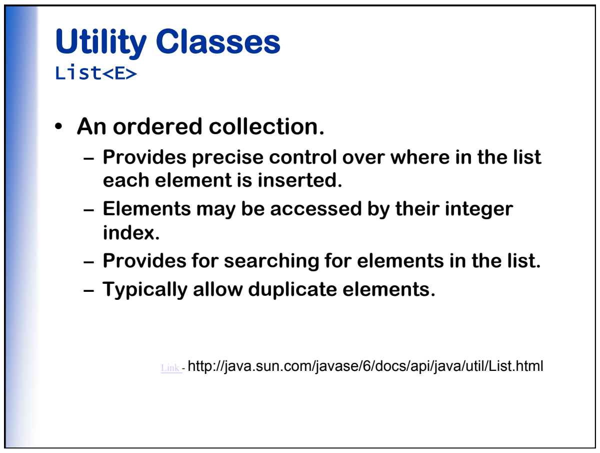
```
boolean hasNext()
E next()
void remove()
```

Link - http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html

Utility Classes
`Collection<E>`

- The `Collection` interface represents collections in a general way
  - Serves as a base interface from which more restrictive collections are extended.

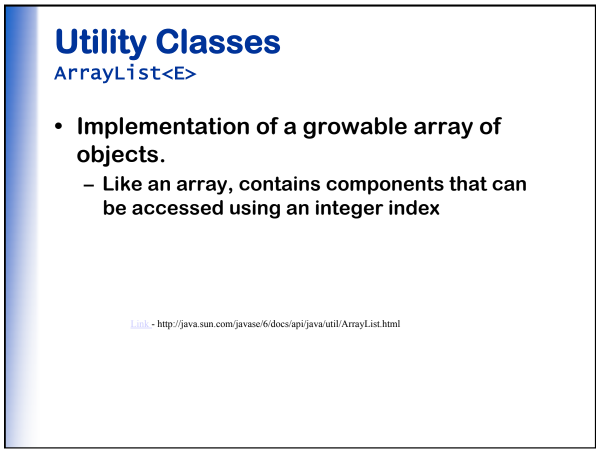
```
java.util.Collection
├ java.util.List
└ java.util.Set
   └ java.util.SortedSet
```



Utility Classes
`List<E>`

- An ordered collection.
  - Provides precise control over where in the list each element is inserted.
  - Elements may be accessed by their integer index.
  - Provides for searching for elements in the list.
  - Typically allow duplicate elements.

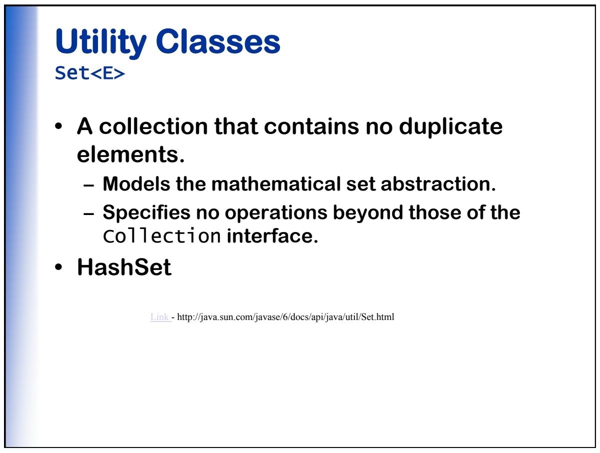
Link - http://java.sun.com/javase/6/docs/api/java/util/List.html



Utility Classes
`ArrayList<E>`

- Implementation of a growable array of objects.
  - Like an array, contains components that can be accessed using an integer index

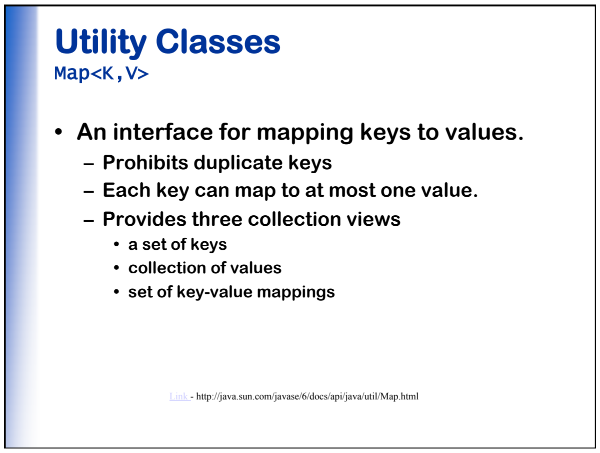
Link - http://java.sun.com/javase/6/docs/api/java/util/ArrayList.html



Utility Classes
`Set<E>`

- A collection that contains no duplicate elements.
  - Models the mathematical set abstraction.
  - Specifies no operations beyond those of the `Collection` interface.
- HashSet

Link - http://java.sun.com/javase/6/docs/api/java/util/Set.html

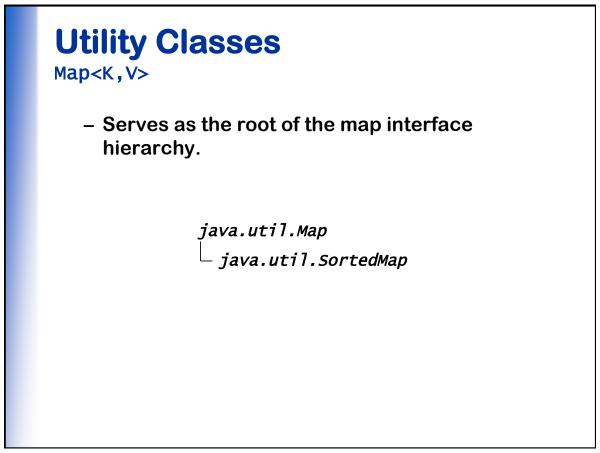


Utility Classes

`Map<K,V>`

- An interface for mapping keys to values.
  - Prohibits duplicate keys
  - Each key can map to at most one value.
  - Provides three collection views
    - a set of keys
    - collection of values
    - set of key-value mappings

Link - http://java.sun.com/javase/6/docs/api/java/util/Map.html

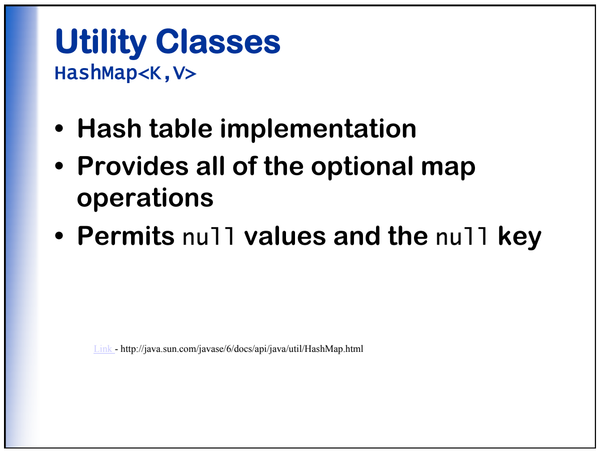


Utility Classes

`Map<K,V>`

- Serves as the root of the map interface hierarchy.

```
java.util.Map
└ java.util.SortedMap
```



Utility Classes

`HashMap<K,V>`

- Hash table implementation
- Provides all of the optional map operations
- Permits `null` values and the `null` key

Link - http://java.sun.com/javase/6/docs/api/java/util/HashMap.html

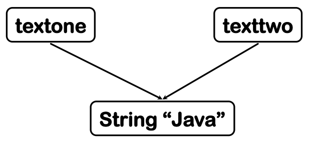
String and StringBuffer

Working with `String`

- `Strings` are immutable, many operations return a new `String`
- The equality operator , ==, tests object references
- Nearly always want to use the equals method, returns true for equivalent `Strings`

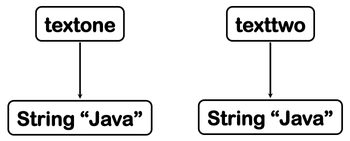
The difference between == and equals() (2)

```
String textone = "Java";
String texttwo = textone;
result = (textone == texttwo);
// result is true
```



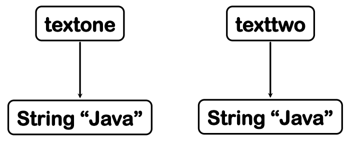
The difference between == and equals() (3)

```
String textone = "Java";
String texttwo = "Java";
result = textone.equals(texttwo);
//result is true
```



The difference between == and equals() (4)

```
String textone = "Java";
String texttwo = new String("Java");
result = (textone == texttwo);
//result is false
```



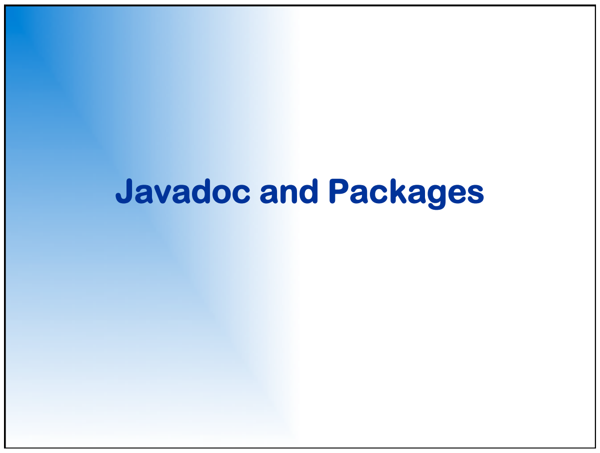
`StringBuffer/StringBuilder`

- Mutable sequence of characters
- More efficient that manipulating `String` objects
- `StringBuffer` is thread-safe

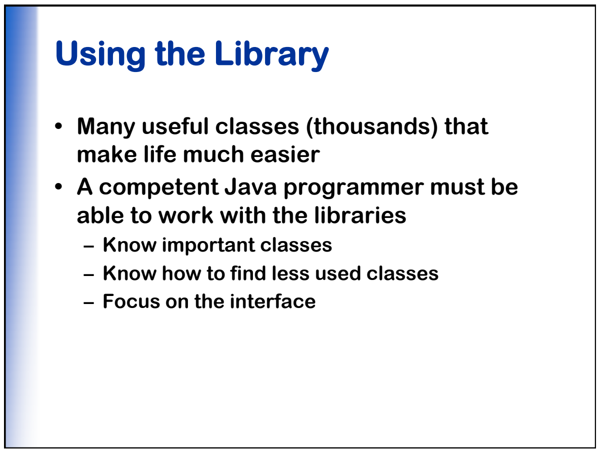
  SB append(*type* t)
  SB insert(int pos, *type* t)
  void setCharAt(int pos, char c)
  String toString()

Link - http://java.sun.com/javase/6/docs/api/java/lang/StringBuffer.html

Link - http://java.sun.com/javase/6/docs/api/java/lang/StringBuilder.html

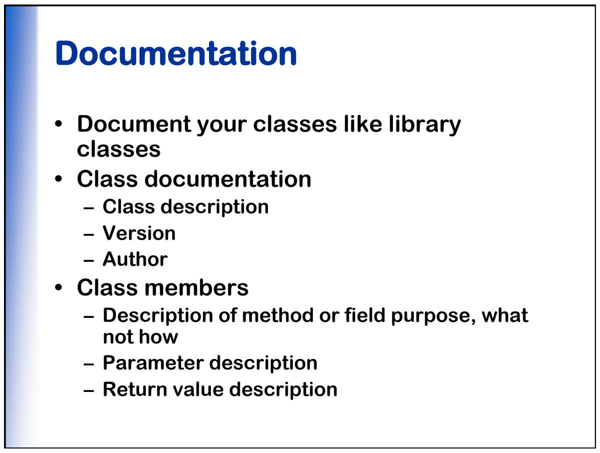


Javadoc and Packages



Using the Library

- Many useful classes (thousands) that make life much easier
- A competent Java programmer must be able to work with the libraries
  - Know important classes
  - Know how to find less used classes
  - Focus on the interface



Documentation

- Document your classes like library classes
- Class documentation
  - Class description
  - Version
  - Author
- Class members
  - Description of method or field purpose, what not how
  - Parameter description
  - Return value description

javadoc

```
/**
 * Simulates the Acme Launcher.
 *
 * @author     Wiley Coyote
 * @version    1.0
 */
public class AcmeLauncher {
    ...
}
```

javadoc

```
/**
 * Launches an Acme rocket at the specified angle and
 * acceleration rate.  The rocket commonly explodes
 * during launch.
 *
 * @param launchAngle the angle the launch at
 * @param accelRate the rate of acceleration when launched
 * @return true if successful, usually false
 */
public boolean launch(double launchAngle,
                      double accelRate) {
    ...
}
```

Packages

- Related classes may be grouped together in packages
- Organizes a number of "`.class`" files, and other packages in a directory
- The package name and its directory share the same name
- Affect class and variable visibility

Packages
Syntax

- The `package` statement must be the first statement in the file. Has the syntax:

  `package` *`package.name`*`;`

  - *`package_name`* describes a package hierarchy, each level separated by a period
  - The directory structure being used must match the package declaration
  - The package name is concatenated with the class name and stored as the full name of the class, again delimited by a period
  - By convention package names are all lowercase

Packages
Accessibility

- Default visibility allows access to all elements within the same package
- Access to elements outside the package may be referenced by:
  - Full package specification or using the `import` statement

Packages
`import`

- Imported names can be used without qualification
- The import statement takes two forms:

  `import` *`package.name.classname`*`;`

  - Allows access to a single class in the package

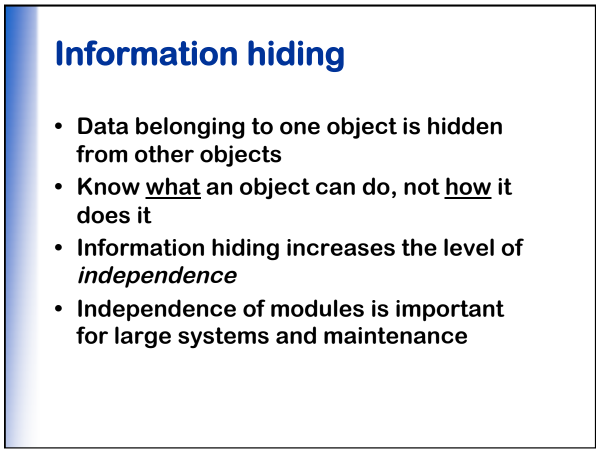
  `import` *`package.name`*`.*;`

  - Allows access to all classes in the package

  `import static` *`package.name.classname.member`*`;`

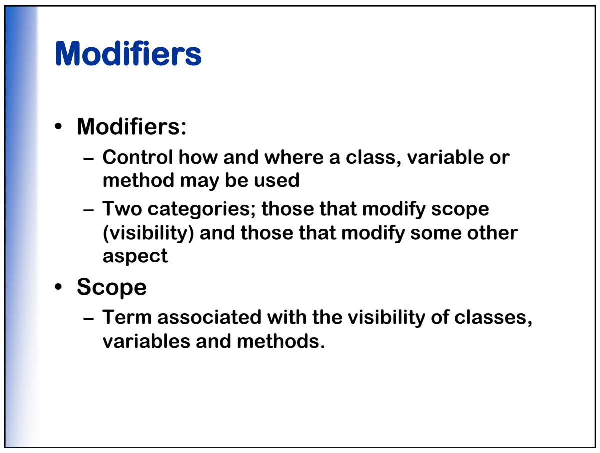
  - Allow import of static methods and variables

Using Access Specifiers



Information hiding

- Data belonging to one object is hidden from other objects
- Know what an object can do, not how it does it
- Information hiding increases the level of *independence*
- Independence of modules is important for large systems and maintenance



Modifiers

- Modifiers:
  - Control how and where a class, variable or method may be used
  - Two categories; those that modify scope (visibility) and those that modify some other aspect
- Scope
  - Term associated with the visibility of classes, variables and methods.

Modifiers
Visibility

Modifier	Element	Meaning
`default` `(package)`	`class` `interface` `method` `variable`	Only accessible within its package
`public`	`class` `interface` `method` `variable`	Accessible anywhere its package is. Only one `public` class is allowed per source file.
`protected`	`method` `variable`	Accessible within its package, and any subclasses.
`private`	`method` `variable`	Accessible within the defining class.

Modifiers
Usage

Modifier	Element	Meaning
`abstract`	`class`	Class cannot be instantiated.
	`interface`	Optional, all interfaces are abstract.
	`method`	No body is provided for the method.
`final`	`class`	Class cannot be subclassed.
	`method`	Method may not be overridden.
	`variable`	Variables value may not be changed.
`static`	`class`	A top-level class, visible outside enclosing class.
	`method` `variable`	A class member. There is only one instance of the member.

Modifiers
Usage (cont.)

Modifier	Element	Meaning
`synchronized`	`method`	Only one thread may execute within the method for a given object at a time.
`transient`	`variable`	Variable is not part of the persistent state of the object.
`native`	`method`	The method is implemented in C, or some other platform-dependent way. No body is provided.

Using Modifiers

- Fields
  - All fields should be private
  - Constants (an exception)

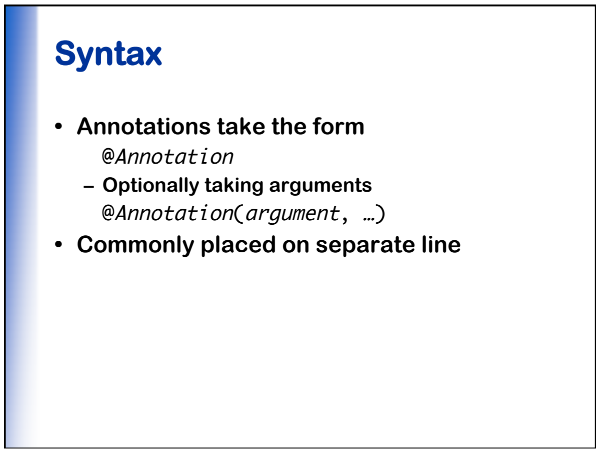
```
public static final ONE_SECOND = 1000;
```

- Methods
  - API methods should be `public`
  - Internal use methods `private`

Annotations

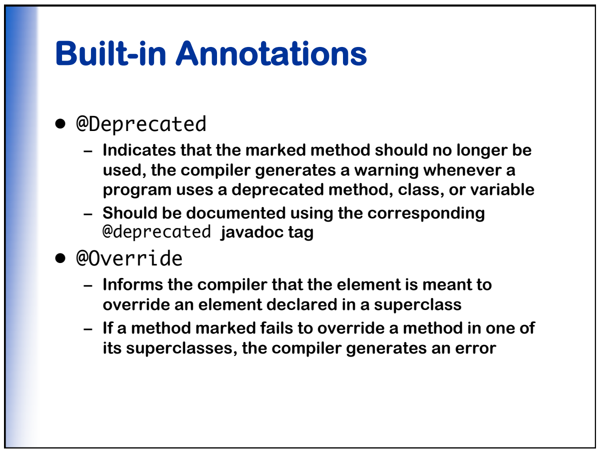
Annotations

- Metadata about your program
  - Provides information for tools
  - Information may be available at run-time
    - Use reflection to access this information
- May be applied to:
  - Types (classes, interfaces, …)
  - Methods & Constructors
  - Fields
  - Variables
  - Parameters
  - Package
  - Annotation types



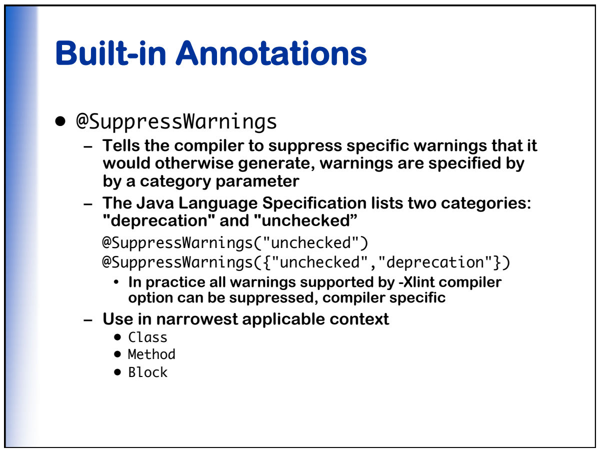
Syntax

- Annotations take the form
  `@Annotation`
  - Optionally taking arguments
    `@Annotation(argument, …)`
- Commonly placed on separate line



Built-in Annotations

- `@Deprecated`
  - Indicates that the marked method should no longer be used, the compiler generates a warning whenever a program uses a deprecated method, class, or variable
  - Should be documented using the corresponding `@deprecated` javadoc tag
- `@Override`
  - Informs the compiler that the element is meant to override an element declared in a superclass
  - If a method marked fails to override a method in one of its superclasses, the compiler generates an error



Built-in Annotations

- `@SuppressWarnings`
  - Tells the compiler to suppress specific warnings that it would otherwise generate, warnings are specified by by a category parameter
  - The Java Language Specification lists two categories: "deprecation" and "unchecked"

    ```
    @SuppressWarnings("unchecked")
    @SuppressWarnings({"unchecked","deprecation"})
    ```
    - In practice all warnings supported by -Xlint compiler option can be suppressed, compiler specific
  - Use in narrowest applicable context
    - `Class`
    - `Method`
    - `Block`

Applications and Testing

Applications
main()

- All applications include a `main()`method
  - Small, create objects and kick things off
- The interpreter invokes the `main()` method of the class specified.
- The `main()` method has a specific signature:

```
public static void main( String[] args ) {}
```

- Applications may return a status to the O/S:

```
System.exit(statusCode);
```

Finding Errors

- Early errors are usually *syntax errors*
  - The compiler will find these
- Later errors are usually *logic errors*, bugs
  - The compiler cannot help with these
  - Testing is the only hope for finding these

Testing fundamentals

- Understand what the unit should do – its *contract*.
  - You will be looking for violations
  - Use positive tests and negative tests
- Test *boundaries*.
  - Zero, One, Full.
    - Search an empty collection.
    - Add to a full collection.

JUnit Testing Framework

- Tests at the method level
- Intended to test the correct functioning of a class's methods
- Extension to provide branch coverage analysis

Concepts

- Test Case
  - A test fixture for executing a set of tests
  - `Setup/Down Annotations`
    - @BeforeClass - run before any test has been executed
    - @Before – run before each test.
    - @After – run after each test
    - @AfterClass – run after all the tests have been executed
  - Methods of the test case defining the tests
    - @Test Annotation

```
        @Test void testMethodName()
```

Concepts

- Assertions
  - Not the Java `assert`
  - Methods of the `Assert` class – Static import

```
assertEquals(type expected, type actual)
assertEquals(String message,
             type expected, type actual)
assertTrue(boolean condition)
assertFalse(boolean condition)
assertSame(Object expected, Object actual)
assertNotSame(Object expected, Object actual)
assertNull(Object obj)
fail(String message)
```

Creating JUnit Tests

- Create a `ClassNameTest` class for each of your classes
- Implement `@Before` and `@After if needed`
- Implement a @Test `testMethodName()` method for each public method of your class
  - Use assertion to evaluate object state
- Eclipse and other tools can generate the `Junit Testcase` class

Debugger

- Print statements
  - Need to be able to disable
  - Logging is a better option
- Debuggers have powerful features
  - Breakpoints
  - Step over, step into, step out
  - Expression examination
  - Watch variables