APPENDIX C

Quick Reference to Java

C.1 VARIABLES

Variables allow us to associate names with values. In Java you must declare a variable before you use it. To declare a variable you specify a type and a name followed by a semicolon.

```
type name;
```

You can also set the value of the variable when you declare it to the result of an expression.

```
type name = expression;
```

The type can be any of the primitive types (`int`, `boolean`, `byte`, `char`, `double`, `float`, `long`, `short`), a class name, or an interface name. The convention is to start variable names with a lowercase letter and uppercase the first letter of each additional word.

```
> int i;
> double totalBill = 32.43 + 20 * 32.43;
> String name = "Mark Guzdial";
> Picture pictureObj;
> List studentList = null;
```

Variable names can be made up of letters, digits, underscores, or currency symbols. They can start with any of these except a digit. Variables can be any word *except* the *reserved words*. The reserved words are:

abstract	assert	boolean	break	byte
case	catch	char	class	const (unused)
continue	default	do	double	else
enum	extends	false	final	finally

float	for	goto (unused)	if	implements
import	instanceof	int	interface	long
native	new	null	package	private
protected	public	return	short	static
strictfp	super	switch	synchronized	this
throw	throws	transient	true	try
void	volatile	while		

Most of the reserved words are also keywords. The only ones that are *not* keywords are `null`, `true`, and `false`. All of the Java reserved words have only lowercase letters.

We can use `System.out.print` or `System.out.println` to print the value of a variable. The second one will also force a new line after the value has printed.

```
> int x = 10
> System.out.println(x);
10
> String name = "Barbara Ericson";
> System.out.println(name);
Barbara Ericson
```

C.2 METHOD DECLARATIONS

Declare a method by specifying the visibility, the return type, the method name, and a parameter list. Method declarations are usually followed by code inside of curly braces which are the statements that will be executed when the method is invoked.

```
visibility returnType name(parameterList)
{
  // statements in the method
}
```

The parameter list is a comma-separated list of the parameters that will be passed to the method. For each parameter, specify a type and a name. Parameters are passed by value, which means that a copy of the value is passed to the method. So primitive variables passed to a method will not be affected by changes in the method after the return from the method, but because the value of an object variable is a reference to the object, a method can change the passed object, and such changes are preserved after the return from the method.

```
public void changeRedAndGreen(double redMult,
                              double greenMult)
```

The convention is to start a method name with lowercase letters and uppercase the first letter of each additional word. A method can return a value by using the `return` statement. The type of the value being returned must match the specified return type.

```
/**
 * Method to create a new picture by rotating the current
 * picture by the given degrees
 * @param degrees the number of degrees to rotate by
 * @return the resulting picture
 */
public Picture rotate(int degrees)
{
  // create a new picture object big enough to hold the result
  // no matter what the rotation is
  Picture result = new Picture((int)
(Math.ceil(rect.getWidth())),
                               (int)
(Math.ceil(rect.getHeight())));
  // other statements in the method
  return result;
}
```

If a method doesn't return any value, the return type should be `void`.

```
  /**
   * Method to decrease the green in the picture by 30
   */
  public void decreaseGreen()
  {
    // method statements
  }
```

If you want all other classes to be able to invoke a method, make the visibility of the method `public`. If you only want to use a method in the class it is declared in use `private` as the visibility. If you leave off the visibility, then the method can be invoked by all classes in the same package. This is called *package visibility*. You can also use *protected visibility* if you want subclasses to be able to override an inherited method, but be aware that all classes in the same package also have access to the method.

C.3 ITERATION (LOOPS)

If you are using Java 5.0 (1.5) or above you can use a for-each loop. The syntax for a for-each loop is:

```
for (type name : collection)
{
 // statements to execute
```

```
}
```

The type is the type of objects in the collection. The name is the local variable name to use. The collection is anything that holds a collection of objects, such as an array, list, or set. The following is an example of using a for-each loop:

```
// loop through all the samples in the array
for (SoundSample sample : sampleArray)
{
   value = sample.getValue();
  sample.setValue(value * 2);
}
```

If you know how many times a loop should repeat, then use a `for` loop. The syntax for a for loop is:

```
for (initializationArea; continuationTest; changeArea)
{
   // statements in the for loop
}
```

You can declare and initialize local variables in the initialization area. You specify a boolean expression for the continuation test. The loop will continue while the test is true. The change area is where you specify how to change variables after each execution of the loop.

```
// loop through all the pixels
for (int i=0; i < pixelArray.length; i++)
    pixelArray[i].setBlue(0);
```

If you don't know how many times a loop should repeat, then use a `while` loop. The syntax of a while loop is:

```
while (continuationTest)
{
   // statements in the while loop
}
```

The statements in the curly braces will be executed as long as the continuation test is true. Often you will initialize variables before the while loop begins and change them just before the end of the while loop statements. But you can do this in the continuation test.

```
// Loop while there is more data
while((line = reader.readLine()) != null)
{
    // print the current line
    System.out.println(line);
}
```

C.4 CONDITIONALS

An `if` takes a boolean expression and evaluates it. If it's true, the `if`'s block is executed. If it's false, the `else` block is executed, if one exists. If you have more than two possibilities, you can add `else if` for each additional one.

```
// tint the shadows darker
if (redValue < 60)
{
    redValue = redValue * 0.9;
    greenValue = greenValue * 0.9;
    blueValue = blueValue * 0.9;
}
// tint the midtones a light brown
// by reducing the blue
else if (redValue < 190)
{
    blueValue = blueValue * 0.8;
}
// tint the highlights a light yellow
// by reducing the blue
else
{
    blueValue = blueValue * 0.9;
}
```

C.5 OPERATORS

`+, -, *, /, %`	Addition, subtraction, multiplication, division, and modulus (remainder). Order of precedence is algebraic.
`<, >, ==, !=, <=, >=`	Logical operators less-than, greater-than, equal-to, not-equal-to, less-than-or-equal, greater-than-or-equal.
`&&, ||, !`	Logical conjunctives `and`, `or`, and `not`.

C.6 STRING ESCAPES

`\t`	Tab character
`\b`	Backspace
`\n`	New line
`\r`	Return

\uXXXX	Unicode character, hexadecimal XXXX

C.7 CLASSES

Each class is usually defined in a separate file with the same name as the class name followed by '.java'. The convention is to uppercase the first letter of all words in a class name.

The syntax to declare a class is:

```
visibility class Name
{
   // fields, constructors, and methods
}
```

You can also specify the parent class using the `extends` keyword. If no parent class is specified, it will be `java.lang.Object`. A child class inherits public and protected fields and methods.

```
visibility class Name extends ParentName
{
   // fields, constructors, and methods
}
```

A class can also implement several interfaces. The list of interfaces is separated by commas and follows the specification of the parent class if given.

```
visibility class Name extends ParentName
   implements Interface1, Interface2, ...
{
   // fields, constructors, and methods
}
```

Here is an example class declaration. The `Student` class will inherit from the `Object` class.

```
public class Student
{
   // fields, constructors, and methods
}
```

C.8 FIELDS

Object fields are the data or state that each object of a class will have. Class (`static`) fields are in the object that defines the class so there is only one and all objects of the class have access to it. Fields are defined inside of a class definition. The convention is to start field names with a lowercase letter and uppercase the first letter of each additional word.

To declare an object field, use:

```
visibility type name;
```

To declare a field and give it a value, use:

```
visibility type name = expression;
```

The visibility for fields is usually `private` so that an object can protect its data from being directly accessed by code in other classes.

To declare a class field, use:

```
visibility static type name;
```

To declare a constant field, use:

```
public static final type name;
```

C.9 CONSTRUCTORS

Constructors are used to initialize the fields in a newly created object. The syntax for a constructor is

```
visibility ClassName(parameterList)
{
   // statements in the constructor
}
```

Constructors are usually defined with `public` visibility. Notice that they do not have a return type. The name on a constructor must match the class name.

C.10 PACKAGES

The Java classes are grouped into packages. You can use any of classes in the package `java.lang`. If you wish to use classes in packages other than `java.lang`, you can use an `import` statement. Import statements go before the class declaration in a file. You can import all the classes in a package using:

```
import name.*;
```

or you can import just a named class using:

```
import name.ClassName;
```

To import all classes in the `java.awt` package use:

```
import java.awt.*;
```

To import just the class `java.awt.Color` use:

```
import java.awt.Color;
```

If you don't import the package or the class, you can use the full name, which is the package name followed by `'.'` and then the class name (`java.awt.Color`).

Here is a table of some of the packages in Java and the interfaces and classes we have used from each of these packages. The interfaces are shown in italics.

`java.lang`	Basic classes in the language	*Comparable*, Object, String, Math
`java.io`	Classes for input and output	BufferedReader, Buffered-

		Writer, FileReader, File-Writer, Reader, Writer, File
`java.awt`	Classes for drawing	*Paint*, Color, Font, Image, Graphics, Graphics2D
`java.net`	Classes for use with networks	URL
`java.sql`	Classes for use with databases	*Connection*, *State-ment*,*ResultSet*, DriverMan-ager
`java.util`	Utility and collection classes	*Iterator*, *List*, *Map*, Array-List, HashMap, TreeMap