Java Quick Reference Card

CS6501 – Internet Programming – Unit – I – Part - I

Prepared by: B.Bhuvaneswaran / AP (SG) / CSE / REC

Java Buzzwords

Simple	Secure	Portable
Object-oriented	Robust	Multithreaded
Architecture-neutral	Interpreted	High performance
Distributed	Dynamic	

Java Virtual Machine

Provides runtime environment to execute java byte code.

Comments

```
// rest of line
/* multiline comment */
/** documentation comment */
```

Data Types

Integers

Name	Width	Range
long	64	–9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
int	32	–2,147,483,648 to 2,147,483,647
short	16	–32,768 to 32,767
byte	8	–128 to 127

Floating point numbers

Name	Width in Bits	Approximate Range
double	64	4.9e–324 to 1.8e+308
float	32	1.4e–045 to 3.4e+038

Characters	–	char	– 16 bit
Boolean	–	boolean	- true/false

Declaring a Variable

type identifier [=value][,identifier [=value]...] ;

Access Control Modifier

public private protected default

Arrays

One-Dimensional Arrays

```
type var-name[ ];
array-var = new type[size];
```

Class Member Access

	Private	No Modifier	Protected	Public
Same class	Y	Y	Y	Y
Same pack. subclass	N	Y	Y	Y
Same pack. non-subclass	N	Y	Y	Y
Diff. pack. subclass	N	N	Y	Y
Diff. pack. non-subclass	N	N	N	Y

Documentation Comments

Comment Insertion

- Packages
- Public classes and interfaces
- Public and protected methods
- Public and protected fields

Class Comments

```
/** A Card object represents a playing card */
public class Card
{
        . . .
}
```

Method Comments

```
@param variable description
@return description        @throws class description
/** Raises the salary of an employee.
@param byPercent the percentage by which to raise the salary
(e.g. 10 = 10%)
@return the amount of the raise
*/
public double raiseSal(double byPercent)
{
        double raise=salary*byPercent/100;
        salary += raise;
        return raise;
}
```

Field Comments

```
        /**
        * The "Hearts" card suit
        */
        public static final int HEARTS = 1;
```

General Comments

```
        @author name     @version text
        @since text      @deprecated text
        @see reference
```

Comment Extraction

```
javadoc -d docDirectory nameOfPackage
(or)
javadoc -d docDirectory nameOfPackage1 nameOfPackage2...
(or)
javadoc -d docDirectory *.java
```

static

Variables

Instance variables declared as static are, essentially, global variables. When objects of its class are declared, no copy of a static variable is made. Instead, all instances of the class share the same static variable.

Methods

- They can only call other static methods.
- They must only access static data.
- They cannot refer to this or super in any way.

General Form of a static method

```
        static type name(parameter-list) {
                // body of the method
        }
```

this Keyword

this can be used inside any method to refer to the current object.

Instance Variable Hiding

As you know, it is illegal in Java to declare two local variables with the same name inside the same or enclosing scopes. Interestingly, you can have local variables, including formal parameters to methods, which overlap with the names of the class' instance variables. However, when a local variable has the same name as an instance variable, the local variable hides the instance variable.

```
// Use this to resolve name-space collisions.
Box(double width, double height, double depth) {
        this.width = width;
        this.height = height;
        this.depth = depth;
}
```

Garbage Collection

Java handles deallocation automatically. The technique that accomplishes this is called garbage collection.

Multidimensional Arrays

```
int twoD[][] = new int[4][5];
```

Alternative Array Declaration Syntax

```
type[ ] var-name;
int al[] = new int[3];
int[] a2 = new int[3];
char twod1[][] = new char[3][4];
char[][] twod2 = new char[3][4];
int[] smallPrimes = { 2, 3, 5, 7, 11, 13 };
```

Anonymous Array

```
smallPrimes = new int[] { 17,19,23,29,31,37 };
```

is shorthand for

```
int[] anonymous = { 17, 19, 23, 29, 31, 37 };
smallPrimes = anonymous;
```

Array Copying

```
int[] luckyNumbers = smallPrimes;
luckyNumbers[5] = 12;
// now smallPrimes[5] is also 12
```

Classes and Objects

A class is a template for an object, and an object is an instance of a class. An object is an instance of a class.

General Form of a Class

```
class classname {
    type instance-variable1;
    type instance-variable2;
    // ...
    type instance-variableN;

    type methodname1(parameter-list) {
        // body of method
    }
    type methodname2(parameter-list) {
        // body of method
    }
    // ...
    type methodnameN(parameter-list) {
        // body of method
    }
}
```

Declaring Objects

```
class-var = new classname( );

Box mybox = new Box(); (or)
Box mybox; // declare reference to object
mybox = new Box(); // allocate a Box object
```

Methods

General Form of a Method

```
type name(parameter-list) {
    // body of method
}
```

General Form of a return

```
return value;
```

General Form of a finalize( ) Method

```
protected void finalize( )
{
    // finalization code here
}
```

Overloading Methods

In Java it is possible to define two or more methods within the same class that share the same name, as long as their parameter declarations are different. When this is the case, the methods are said to be overloaded, and the process is referred to as method overloading. Method overloading is one of the ways that Java implements polymorphism.

Constructors

A constructor initializes an object immediately upon creation. It has the same name as the class in which it resides and is syntactically similar to a method. Once defined, the constructor is automatically called immediately after the object is created, before the new operator completes. Constructors look a little strange because they have no return type, not even void.

“for each” loop

Sets the given variable to each element of the collection and then executes the statement (which, of course, may be a block). The collection expression must be an array or an object of a class that implements the Iterable interface, such as ArrayList

```
for (variable : collection) statement
```

Strings

Strings Are Immutable

String API

```
char charAt(int index)
int compareTo(String other)
boolean endsWith(String suffix)
boolean equals(Object other)
boolean equalsIgnoreCase(String other)
int indexOf(String str)
int indexOf(String str, int fromIndex)
int indexOf(int cp)
int indexOf(int cp, int fromIndex)
int lastIndexOf(String str)
int lastIndexOf(String str, int fromIndex)
int lastindexOf(int cp)
int lastindexOf(int cp, int fromIndex)
int length()
String replace(CharSequence oldString, CharSequence newString)
boolean startsWith(String prefix)
String substring(int beginIndex)
String substring(int beginIndex, int endIndex)
String toLowerCase()
String toUpperCase()
String trim()
```

Packages

Class Importation

```
java.util.Date today = new java.util.Date();
(or)
import java.util.*;
Date today = new Date();

import java.util.*;
import java.sql.*;
Date today;
// ERROR--java.util.Date or java.sql.Date?

import java.util.*;
import java.sql.*;
import java.util.Date;
java.util.Date deadline = new java.util.Date();
java.sql.Date today = new java.sql.Date(...);
```

Static Imports

```
import static java.lang.System.*;
out.println("Goodbye, World!");
// i.e., System.out
exit(0); // i.e., System.exit
```