Summary of Basic Java Syntax

Philip Chan

November 7, 2006

1 Primitive Types

`byte`
`short`
`int`
`long`
`float`
`double`
`char`
`boolean`

2 Keyboard Input

```
Scanner keyboard     = new Scanner(System.in); // new creates an object
int     intValue     = keyboard.nextInt();     // object.method()
short   shortValue   = keyboard.nextShort();   // similarly for byte, long
float   floatValue   = keyboard.nextFloat();
double  doubleValue  = keyboard.nextDouble();
String  tokenValue   = keyboard.next();        // default delimiters: whitespace
String  lineValue    = keyboard.nextLine();
boolean booleanValue = keyboard.nextBoolean();
```

3 Screen Output

```
System.out.println(...);
System.out.print(...);
```

4 Arithmetic Operators

`+`	addition
`-`	subtraction
`*`	multiplication
`/`	division
`%`	modulo (remainder)
`++var`	pre-increment
`var++`	post-increment
`--var`	pre-decrement
`var--`	post-decrement

5 Assignment Operators

`=`	assignment
`+=`	addition assignment
`-=`	subtraction assignment
`*=`	multiplication assignment
`/=`	division assignment
`%=`	modulo assignment

6 Relational (Comparison) Operators

`<`	less than
`<=`	less than or equal
`>`	greater than
`>=`	greater than or equal
`==`	equal
`!=`	not equal

7 Logical Operators

`&&`	and [short-circuit (lazy) evaluation]
`||`	or [short-circuit (lazy) evaluation]
`!`	not
$\wedge$	exclusive or
`&`	and [complete (eagar) evaluation]
`|`	or [complete (eagar) evaluation]

8 String Class

```
String  stringValue    = "Hello";
String  concatString   = stringValue1 + stringValue2;
int     length         = stringValue.length();
char    charAtIndex    = stringValue.charAt(index);
String  substring      = stringValue.substring(startIndex, endIndex); // excluding endIndex
String  substring      = stringValue.substring(startIndex);
int     startIndex     = stringValue.indexOf(stringToSearchFor);
String  lowerCaseStr   = stringValue.toLowerCase();                   // similarly toUpperCase()
boolean sameContent    = stringValue1.equals(stringValue2);
int     ordering       = stringValue1.compareTo(stringValue2);
```

9 Math Class

`Math.PI`	π (3.14159...)
`Math.E`	e (2.71828...)
`Math.abs(x)`	$\lvert x \rvert$
`Math.ceil(x)`	$\lceil x \rceil$
`Math.log(x)`	$\ln x$ ($\log_e x$)
`Math.log10(x)`	$\log_{10} x$
`Math.pow(x, y)`	x^y
`Math.round(x)`	nearest integer
`Math.sqrt(x)`	$\sqrt{x}$

10 Branching (Conditional) Statements

10.1 Simple `if` statement

```
if (<condition>)
  <statement>;
else     // else part is optional
  <statement>;
```

10.2 Compound `if` statement

```
if (<condition>)
{
    <statement>;
    ...
}
else     // else part is optional
{
    <statement>;
    ...
}
```

10.3 `if-else-if` statement

```
if (<condition1>)
{
    <statement>;
    ...
}
else if (<condition2>)
{
    <statement>;
    ...
}
else  // else part is optional
{
    <statement>;
    ...
}
```

10.4 `switch` statement

```
switch (<intOrCharExpression>)
{
    case <value1>:
        <statement>;
        <...>
        break;
    ...
    case <valueN>:
        <statement>;
        ...
        break;
    default:  // default case is optional
        <statement>;
        ...
}
```

11 Loop Statements

11.1 `while` loop

```
while (<condition>)
{
    <statement>;
    ...
}
```

11.2 do-while loop

```
do
{
    <statement>;
    ...
}
while (<condition>);  // note the semicolon
```

11.3 for loop

```
for (<initialization>; <condition>; <update>)  // "ICU"
{
    <statement>;
    <...>
}
```

12 Classes

12.1 A Basic Class

```
public class <className>
{
    // instance variables (attributes for each object)
    private <type> <attrName>;

    // public methods (for each object)
    public <returnType> <methodName>(<formalParameters>)
        {
            <statement>;
            <...>;

            return <expression>;  // not needed if returnType is void
        }

}
```

12.2 A Class with Various Options

```
public class <className>
{
    // global constants
    public static final <type> <globalConstName>;

    // semi-global constants for the class
    private static final <type> <semiGlobalConstName>;

    // constant attributes for each object
    private final <type> <attrName>;

    // instance variables (attributes for each object)
    private <type> <attrName>;

    // constructor
    public <classname>(<formalParameters>)
        {
            <statement>;
            <...>;
        }
```

```
    // public methods (for each object)
    public <returnType> <methodName>(<formalParameters>)
        {
            <statement>;
            <...>;

            return <expression>;  // not needed if returnType is void
        }


    // private methods (helper methods for each object)
    private <returnType> <methodName>(<formalParameters>)
        {
            <statement>;
            <...>;

            return <expression>;  // not needed if returnType is void
        }


    // public static methods (for the class)
    public static <returnType> <methodName>(<formalParameters>)
        {
            <statement>;
            <...>;

            return <expression>;  // not needed if returnType is void
        }


    // private static methods (helper methods for the class)
    private static <returnType> <methodName>(<formalParameters>)
        {
            <statement>;
            <...>;

            return <expression>;  // not needed if returnType is void
        }
}
```

13 Arrays

13.1 One-dimensional arrays for primitive types

```
<type>[] <arrayName> = new <type>[<length>];
```

13.2 One-dimensional arrays for class types

```
// allocate space for addresses of objects
<classType>[] <arrayName> = new <classType>[<length>];

// create an object for each array element, usually in a loop
<arrayName>[<index>] = new <classType>();
```

13.3 Two-dimensional arrays for primitive types

```
<type>[][] <arrayName> = new <type>[<rowLength>][<columnLength>];
```

13.4 Two-dimensional arrays for class types

```
// allocate space for addresses of objects
<classType>[][] <arrayName> = new <classType>[<rowLength>][<columnLength>];

// create an object for each array element, usually in a nested loop
<arrayName>[<rowIndex>][<columnIndex>] = new <classType>();
```

13.5 Ragged two-dimensional arrays

```
<type>[][] <arrayName> = new <type>[<rowLength>][];

// create an array for each row
<arrayName>[<rowIndex>] = new <type>[columnLength];
```