



Java Basics – Part 2

Unit 04



Section Goals

- Continue learning the basic syntax of Java
- Understand how objects are used
- Provide an introduction to built-in Java classes:
 - String & StringBuffer
 - Wrapper classes

2 categories of types in Java

Primitive Types

byte
char
short
int
long
float
double
boolean

initialized to 0

initialized to false

Primitive type instance
Variables initialized by
Default

Primitive types contain exactly one
value of its declared type at a time

Local variables
not initialized by default

versus

Reference Types

all others
ex. String
GradeBook
Scanner

variables of reference types
or references are locations of
objects in memory
Objects that are referenced may
contain many instance
variables and methods

*Instantiating an object and saving the return value
of one of its methods in an integer primitive
instance variable*

```
int myVariable = 0;  
    // an integer primitive called myVariable gets zero  
MyObject obj = new MyObject();  
    // an object called obj of reference type MyObject is instantiated by invoking  
    // its constructor using the reserved word new  
    // the class is defined in a file called MyObject.java  
    // the object in memory is referenced by the identifier obj  
myVariable = obj.getSomething();
```

MyObject lives
In a file on the
Hard Drive

(It is an idea)

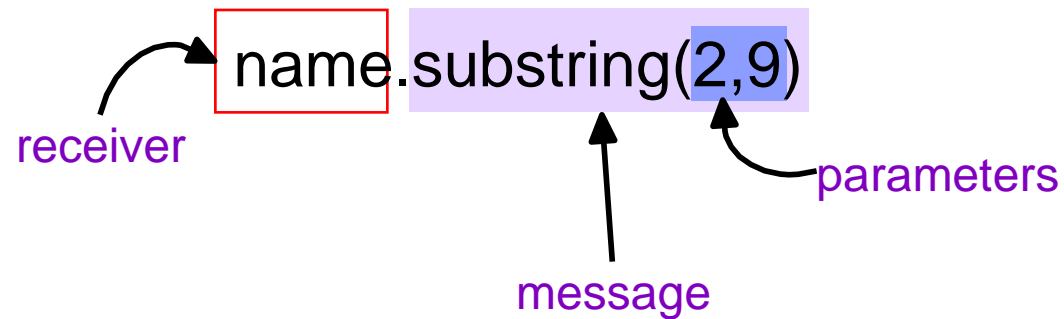
myVariable and obj live in the
computers
Memory (RAM or page file)

(these are real)

Note: there are other ways to write this code, this is one example

Objects and Messages

- Objects provide more complex behavior than primitives
- Objects respond to messages
 - Use the dot "." operator



Declaring and Initializing Objects

- Just like primitives and arrays, objects must be declared before they can be used
 - The declaration requires the *type* of the object
 - We will learn exactly what *types* are later, and how to define them
 - Use '=' for assignment (including initialization)
 - Initialization of an object often uses the `new` operator
 - An object can be initialized to `null`
- Arrays of objects are declared just like arrays of primitives
 - Arrays of objects default to initialization with `null`
- Examples:

```
Employee emp1 = new Employee(123456);
Employee emp2;
emp2 = emp1;
Department dept[] = new Department[100];
Test[] t = {new Test(1), new Test(2)};
```

Identity

- The == operator
 - Tests for exact object identity
 - Checks whether two variables reference the same object
 - For primitive types, checks for equal values

```
Employee a = new Employee(1);  
Employee b = new Employee(1);  
if (a==b)... // false
```

```
Employee a = new Employee(1);  
Employee b = a;  
if (a==b)... // true
```

```
int a = 1;  
int b = 1;  
if (a==b)... // true
```

Wrapper Classes

- Primitives have no associated methods
- Wrapper classes:
 - Encapsulate primitives
 - Provide methods to work on them
 - Are included as part of the base Java API

Primitive Type	Wrapper Class
boolean	Boolean
byte	Byte
char	Character
double	Double
float	Float
int	Integer
long	Long
short	Short

Using Wrapper Classes

```
double number = Double.parseDouble("42.76");
```

```
String hex = Integer.toHexString(42);
```

```
double value = new Integer("1234").doubleValue();
```

```
String input = "test 1-2-3";
int output = 0;
for (int index = 0; index < input.length(); index++) {
    char c = input.charAt(index);
    if (Character.isDigit(c))
        output = output * 10 + Character.digit(c, 10);
}
System.out.println(output) // 123
```

Strings

- Any number of characters between double quotes is a String:

```
String a = "A String";  
String b = "";
```

- String can be initialized in other ways:

```
String c = new String();  
String d = new String("Another String");  
String e = String.valueOf(1.23);  
String f = null;
```

Concatenating Strings

- The + operator concatenates Strings:

```
String a = "This" + " is a " + "String";
```

- Primitive types used in a call to `println` are automatically converted to Strings

```
System.out.println("answer = " + 1 + 2 + 3);  
System.out.println("answer = " + (1+2+3));
```

Do you get the same output from the above examples?

Comparing Strings

- `oneString.equals(anotherString)`
 - Tests for equivalence
 - Returns `true` or `false`
- `oneString.equalsIgnoreCase(anotherString)`
 - Case insensitive test for equivalence
 - Returns `true` or `false`
- `oneString == anotherString` is problematic

```
String name = "Joe";  
if ("Joe".equals(name))  
    name += " Smith";
```

```
boolean same = "Joe".equalsIgnoreCase("joe");
```

String Messages

- Strings are objects; objects respond to messages
 - Use the dot (.) operator to send a message
 - String is a class, with methods (more later)

```
String name = "Joe Smith";
name.toLowerCase(); // "joe smith"
name.toUpperCase(); // "JOE SMITH"
" Joe Smith ".trim(); // "Joe Smith"
"Joe Smith".indexOf('e'); // 2
"Joe Smith".length(); // 9
"Joe Smith".charAt(5); // 'm'
"Joe Smith".substring(5); // "mith"
"Joe Smith".substring(2,5); // "e S"
```

StringBuffer

- `StringBuffer` is a more efficient mechanism for building strings
 - String concatenation
 - Can get very expensive
 - Is converted by most compilers into a `StringBuffer` implementation
 - If building a simple `String`, just concatenate
 - If building a `String` through a loop, use a `StringBuffer`

```
StringBuffer buffer = new StringBuffer(15);
buffer.append("This is ");
buffer.append("String");
buffer.insert(7, " a");
buffer.append('.');
System.out.println(buffer.length()); // 17
System.out.println(buffer.capacity()); // 32
String output = buffer.toString();
System.out.println(output); // "This is a String."
```

Conditional Statement Types: if-else

- An if-else statement is a conditional expression that must return a boolean value
- `else` clause is optional
- Braces are not needed for single statements but highly recommended for clarity

```
if (x > 10) {  
    if (x != 20) {  
        System.out.println("x is not 20");  
    }  
    else {  
        System.out.println("x = " + x);  
    }  
}  
else {  
    System.out.println("x is less than 11");  
}
```

Conditional Statement Types: switch

- Switch statements test a single variable for several alternative values
- Cases without break will “fall through” (next case will execute)
- default clause handles values not explicitly handled by a case

```
switch (day) {  
  case 0:  
  case 1:  
    rule = "weekend";  
    break;  
  case 2:  
    ...  
  case 6:  
    rule = "weekday";  
    break;  
  default:  
    rule = "error";  
}
```

```
if (day == 0 || day == 1) {  
  rule = "weekend";  
} else if (day > 1 && day < 7) {  
  rule = "weekday";  
} else {  
  rule = error;  
}
```

The Ternary Operator

- Shortcut for if-else statement:
(`<boolean-expr> ? <true-choice> : <false-choice>`)
- Can result in shorter code
 - Make sure code is still readable

```
if (x>LIMIT) {  
    warning = "Too Big";  
} else {  
    warning = null;  
}
```

VS.

```
warning = (x>LIMIT) ? "Too Big" : null ;
```

Looping Statement Types: while

- Executes a statement or block as long as the condition remains true
- while () executes zero or more times'
- do...while() executes at least once.

```
int x = 2;
while (x < 2) {
    x++;
    System.out.println(x);
}
```

```
int x = 2;
do {
    x++;
    System.out.println(x);
} while (x < 2);
```

Looping Statement Types: for

- A `for` loop executes the statement or block `{ }` which follows it
 - Evaluates "start expression" once
 - Continues as long as the "test expression" is true
 - Evaluates "increment expression" after each iteration
- A variable can be declared in the `for` statement
 - Typically used to declare a "counter" variable
 - Typically declared in the "start" expression
 - Its scope is restricted to the loop

```
for (start expr; test expr; increment expr) {  
    // code to execute repeatedly  
}
```

```
for (int index = 0; index < 10; index++) {  
    System.out.println(index);  
}
```

for vs. while

- These statements provide equivalent functionality
 - Each can be implemented in terms of the other
- Used in different situations
 - `while` tends to be used for open-ended looping
 - `for` tends to be used for looping over a fixed number of iterations

```
int sum = 0;
for (int index = 1; index <= 10; index++) {
    sum += index;
}
```

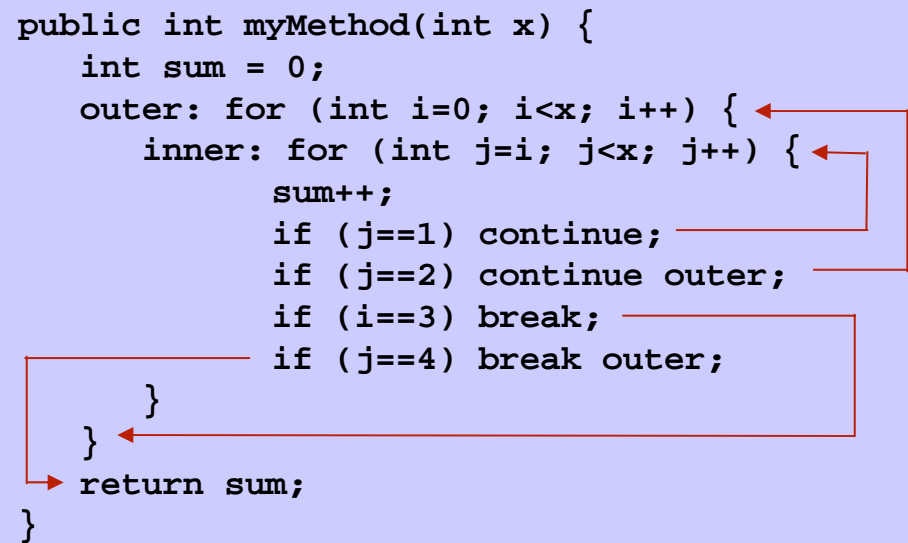
```
int sum = 0;
int index = 1;
while (index <= 10) {
    sum += index;
    index++;
}
```

Branching Statements

- `break`
 - Can be used outside of a switch statement
 - Terminates a for, while or do-while loop
 - Two forms:
 - Labeled: execution continues at next statement outside the loop
 - Unlabeled: execution continues at next statement after labeled loop
- `continue`
 - Like break, but merely finishes this round of the loop
 - Labeled and unlabeled form
- `return`
 - Exits the current method
 - May include an expression to be returned
 - Type must match method's return type
 - Return type "void" means no value can be returned

Sample Branching Statements

```
public int myMethod(int x) {  
    int sum = 0;  
    outer: for (int i=0; i<x; i++) {  
        inner: for (int j=i; j<x; j++) {  
            sum++;  
            if (j==1) continue;   
            if (j==2) continue outer;   
            if (i==3) break;   
            if (j==4) break outer;   
        }  
    }  
    return sum;  
}
```



Scope

- A variable's scope is the region of a program within which the variable can be referred to

– Variables declared in:

- Methods can only be accessed in that method
- A Loop or a block can only be accessed in that loop or block

```
int a = 1;
for (int b = 0; b < 3; b++){
    int c = 1;
    for (int d = 0; d < 3; d++){
        if (c < 3) c++;
    }
    System.out.print(c);
    System.out.println(b);
}
x a = c; // ERROR! c is out of scope
```

abcd
abc
a

Homework #4

1) Find the error(s) in the following method and write a solution:

```
public static void main(String[] args){
    double a;
    double b = 6.2;
    double c = a + b;
    int d = 0.5;
    int e = 7;
    int f = d + e;
    Sytem.out.println("c = " + c);
    Sytem.out.println("f = " + f);
}
```

Test your code in Eclipse to ensure its correctness.

2) Write the output of the following program:

```
public class Test{

    public static void main(String[] args){
        int[] array1 = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18};
        System.out.println(array1[0]);
        System.out.println(array1[array1[2]] );
        System.out.println(array1[4 + array1[1]] );
        System.out.println(array1[array1[array1[2]]]);
        System.out.println(array1[8] + 5 + " ");
    }
}
```

Homework #4

3. What are the four types of Java integers?
4. What are the two types of Java floating point numbers?
5. Using our conventions, HelloWorld is an identifier for what?
6. When does Java provide implicit casts?
7. long has how many bytes?
8. List four of the five kinds of literals:
9. `/* Is this
a javadoc
comment? */`
10. Do problem 4.22

What You Have Learned

•In this unit, you have learned about the basics of Java syntax, including:

–Object declaration and initialization

–Specific classes

- String & String Buffer
- Wrapper classes

–Java statement syntax

–Scope