Java & JEE Training

Day 2 – Elements of Java programming
Deep dive into Java programming

MindsMapped Consulting

Agenda

- Quick review of some important concepts from last class
  - History of Java
  - JDK and JRE
  - Byte Code and JVM (Java Virtual Machine)
  - Platform Independence
- Principles of Object Oriented Programming
- Writing your first Java Application
- Elements of Java programming language
  - Built in Data Types
  - Conditional Statements
  - Loops

History of Java - Milestones

- 1990→ Sun Microsystems decided to develop special software that could be used to manipulate consumer electronic devices. A team of Sun Microsystems programmers headed by James Gosling was formed to undertake this task.
- 1991→ After exploring the possibility of most Object Oriented Programming Language C++, the team announced a new language named "Oak".
- 1992→ The team, known as a Green Project team by Sun, demonstrated the application of their new language to control a list of home appliances using a hand-held device with a tiny touch sensitive screen.
- 1993→ The World Wide Web(WWW) appeared on the internet and transformed the text-based Internet into a Graphical-rich environment. The green Project team came up with the idea of developing Web Applets(tiny programs) using the new language that could run on all types of computers connected to Internet.
- 1994→ The team developed a web browser called "Hot Java" to locate and run applet programs on Internet. Hot Java demonstrated the power of the new language, thus making it instantly popular among the Internet users.
- 1995→ Oak was named "Java", due to some legal snags. Java is just a name and is not an acronym. Many popular companies including Netscape and Microsoft announce to their support to Java.
- 1996→ Java established itself not only a leader for Internet Programming but also as a general-purpose, object oriented programming language. Java found its home.

Java - Features

The most striking feature of the language is that it is a platform-neutral language. Java is a first programming language that is not tied to any particular hardware or operating system.

Features of Java :

- **Compiled and Interpreted.**
- **Platform-Independent and Portable**
- **Object-Oriented**
- **Robust and Secure**
- **Distributed**
- **Familiar, Simple and Small**
- **Multithreaded and Interactive**
- **High Performance**
- **Dynamic and Extensible**

JDK & JRE

Java Environment includes a large number of development tools and hundreds of classes and methods. The development tools are part of the system known as Java Development Kit(JDK) and the classes and methods are part of the Java Standard Library (JSL), also known as the Application Programming Interface (API).

JDK : Java Development Kit comes with a collection of tools that are used for developing and running Java Programs. They include :

- **appletviewer->** Enables us to run Java Applets (Without using java compatible browser)
- **java→** Java Interpreter, which runs applets and applications by reading and interpreting bytecode files.
- **Javac→** The Java compiler, which translates Java source code to byte code files that the interpreter understand.
- **Javadoc→** Creates HTML-format documentation from Java source code files.
- **Javah→** Produces header files for use with native methods.
- **javap→** Java disassembler, which enables us to convert bytecode files into a program description.
- **Jdb→** Java debugger, which helps us to find errors in our programs.

Byte Code & JVM(Java Virtual Machine)

Since platform-independence is a defining characteristic of Java, it is important to understand how it is achieved. Programs exist in two forms; source code and object code. Source Code is the textual version of the program that you write using a text editor. The programs printed in a book are shown as source code. The executable form of a program is object code. The computer can execute object code. Typically, object code is specific to a particular CPU. Therefore, it cannot be executed on a different platform. Java removes this feature in a very elegant manner.

Like all computer languages, a java program begins with its source code. The difference is what happens when a Java program is compiled. Instead of producing executable code, the Java Compiler produces an object file that contains bytecode. Bytecodes are instructions that are not for any specific CPU. Instead, they are designed to be interpreted by a Java Virtual Machine (JVM). The key to Java's platform-independence comes from the fact that the same bytecodes can be executed by any JVM on any platform. As long as there is a JVM implemented for a given environment, it can run any Java program. For example, Java programs can execute under Windows 98,Solaris,IRIX, or any other platform for which a JVM can be implemented for that platform. This would then allow any Java program to execute in that new environment.

Platform Independent

Compilation is the process of converting the code that you type into a language that the computer understands-machine language. When you compile a program, the compiler checks for syntactical errors in code and lists all the errors on the screen. You have to rectify the errors and recompile the program to get the machine language code. The Java compiler compiles the code to bytecode that is understood by the Java environment.

The bytecode is the result of compiling a Java program. You can execute this code on any platform. In other words, due to the bytecode compilation process and interpretation by a browser, Java programs can be executed on a variety of hardware and operating systems. The only requirement is that the system should have a java-enabled Internet browser. The java compiler is written in Java, and the interpreter is written in C. The Java Interpreter can execute Java Code directly on any machine on which a Java interpreter has been installed.

Java Compiler and Interpreter

Stage 1: Process of Compilation

Stage 2: Process of Converting bytecode into machine code

Introduction to Object Oriented Programming

MindsMapped Consulting

OOP vs Procedural programming

- Procedural programming focuses on creating functions, whereas OOP focuses on both data and functions.
- Procedural programming separates data of the program from the operation that it performs, whereas in OOP data and function are tied together in a single unit known as class.

Pillars of OOP

Principles of Object Oriented Programming

Object Oriented Programming (OOP) attempts to **emulate the real world** in software systems.

The real world consists of objects, categorized in classes. In Object Oriented Programming, classes have attributes, represented by data member. The attributes distinguish an object of the class. Classes have behaviors, which are represented by methods. The methods define how an object acts or reacts.

Features of Object Oriented Programming :

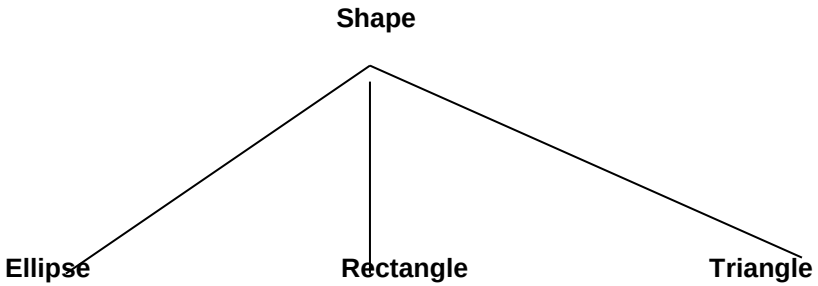
Encapsulation(Hiding) [Security] - Objects provide the benefit of information hiding. Electrical wiring in a television should not be tempered with, and therefore should be hidden from the user. Object Oriented programming allows you to encapsulate data that you do not want users of the object to access. Typically, attributes of a class are encapsulated.

Abstraction - Abstraction allows us to focus on only those parts of an object that concern us. Person operating the television does not need to know the intricacies of how it works. The person just needs to know how to switch it on, change channels, and adjust the volume. All the details that are unnecessary to users are encapsulated, leaving only a simple interface to interact with. Providing users with only what they need to know is known as abstraction. i.e. Abstraction lets us ignore the irrelevant details and concentrate on the essentials.

Principles of Object Oriented Programming

Inheritance [Reusability]:- Inheritance is the process by which objects of one class acquire the properties of objects of another class. Inheritance supports the concept of hierarchical classification. In OOP, the concept of inheritance provides the idea of reusability. This means that we can add additional features to an existing class without modifying it. This is possible by deriving a new class from the existing one. The new class will have the combined features of both the classes.

Polymorphism [Flexibility]:- Polymorphism means "One Interface, multiple implementations."



The class Shape defines a method called getArea() that returns the area of a shape. However, this method is not implemented by that class. Therefore, it is an abstract method and Shape is an abstract class.

This means that no objects of class Shape can be created. However, the functionality of that class can be inherited by its subclass. The various subclasses of Shape like Ellipse, Rectangle, Triangle do implement the getArea() method.

Writing your first Java application – Hello World

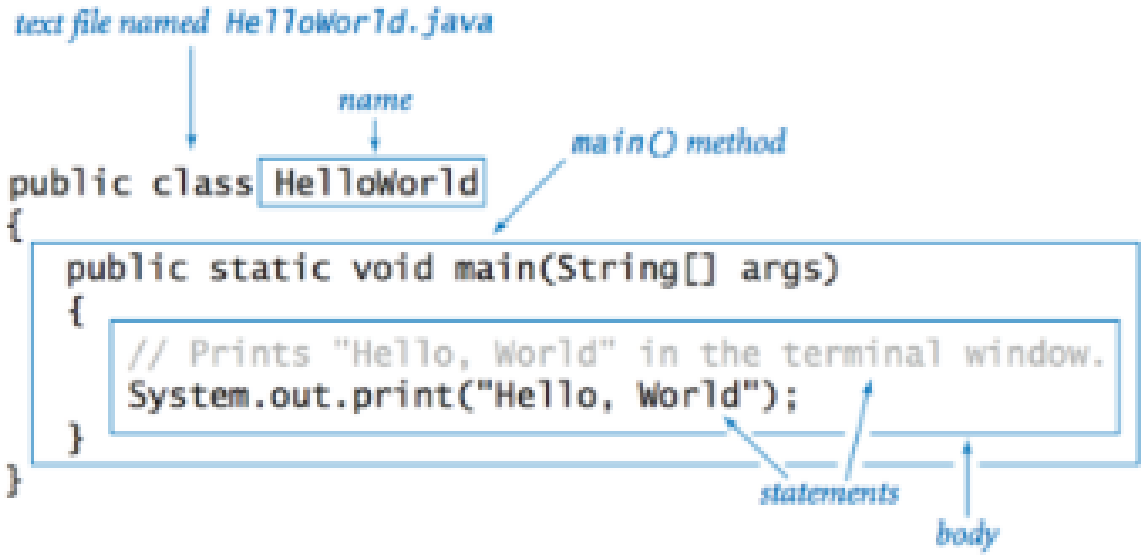
MindsMapped Consulting

First Java Application

Create the File

The first step to create the HelloWorld application is to copy the text below into a file called HelloWorld.java using your favorite text editor.

It is very important to call the file HelloWorld.java, because the compiler expects the file name to match the class identifier



First Java Application

Class Declaration

The first line public class HelloWorld declares a class, which is an Object-Oriented construct. As stated earlier Java is true Object-Oriented language and therefore, everything must be placed inside a class. Class is a keyword and declares that a new class definition follows.

Opening Brace

Every class definition in Java begins with an opening brace "{" and ends with a matching closing brace "}", appearing in the last line in the example.

The main() method

Every java application program must include the main() method. This is starting point for the interpreter to begin the execution of the program. A Java application can have any number of classes but only one of them must include a main method to initiate the execution.

Public : The Keyword public is an access specifier that declares the main method as unprotected and therefore making it to accessible to all other classes.

Static : The keyword static which declares this method as one that belongs to the entire Class and not a part of any Objects of the class.

First Java Application

The main must always be declared be declared as static since the interpreter uses this method before any objects are created.

Void : The type modifier void states that the main method does not return any value.

All parameters to a method are declared inside a pair of parentheses. Here,

String args[] declares a parameter named args, which contains an array of objects

Of the class type String.

The Output Line

The only executable Statement in the program is

```
System.out.println("Hello World!!");
```

Since Java is a true Object Oriented Language, every method must be part of an Object. The println method is a member of the out Object, which is static data Member of the System class. This line prints

```
Hello World!!
```

to the screen. The method println always appends a newline character to the end of the string.

First Java Application

Compile the Code
To compile the program, you need to first install the JDK. Then, use the program javac included with the JDK to convert the text to code which the computer can run. To run javac, on a Macintosh drag the source file over the javac icon. On any other computer, type the line:
at a command prompt. `javac HelloWorld.java`

The javac program creates a file called HelloWorld.class from the HelloWorld.java file. Inside this file (HelloWorld.class) is text known as *bytecodes* which can be run by the Java interpreter.

Run the Program
Now that you have compiled the program, you can run it by typing at the command prompt:

```
java HelloWorld
```

After you do this, the computer should print to the screen

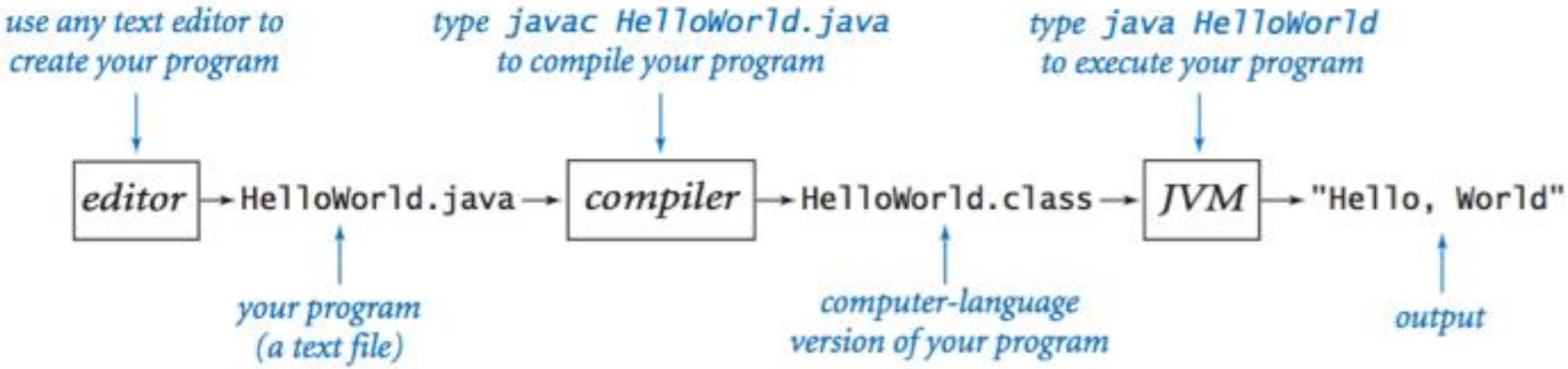
```
Hello World!!
```

That may not seem very interesting, but then it's a simple program. If you don't see the Hello World!! on the screen, go back and make sure you have typed in the file exactly as shown, and make sure that you called the file HelloWorld.java.

Breaking down the “Hello World” Program

MindsMapped Consulting

Java programming process



Exercise

Write a program TenHelloWorlds.java that prints "Hello, World" ten times.

3 Types of Errors in Programming

Most errors are easily fixed by carefully examining the program as we create it, in just the same way as we fix spelling and grammatical errors when we type an e-mail message.

- ***Compile-time errors.*** These errors are caught by the system when we compile the program, because they prevent the compiler from doing the translation (so it issues an error message that tries to explain why).

- ***Run-time errors.*** These errors are caught by the system when we execute the program, because the program tries to perform an invalid operation (e.g., division by zero).

- ***Logical errors.*** These errors are (hopefully) caught by the programmer when we execute the program and it produces the wrong answer. Bugs are the bane of a programmer's existence. They can be subtle and very hard to find.

Providing Input to the HelloWorld program

- Typically, we want to provide *input* to our programs: data that they can process to produce a result. The simplest way to provide input data is illustrated.
- Whenever this program is executed, it reads the **command-line argument** that you type after the program name and prints it back out to the terminal as part of the message.

Providing Input to the HelloWorld program

```
/*************************************************************************
 *  Compilation:  javac UseArgument.java
 *  Execution:    java UseArgument yourname
 *
 *  Prints "Hi, Bob. How are you?" where "Bob" is replaced by the
 *  command-line argument.
 *
 *  % java UseArgument Bob
 *  Hi, Bob. How are you?
 *
 *  % java UseArgument Alice
 *  Hi, Alice. How are you?
 *
 *************************************************************************/

public class UseArgument {

    public static void main(String[] args) {
        System.out.print("Hi, ");
        System.out.print(args[0]);
        System.out.println(". How are you?");
    }

}
```

Exercise

Modify UseArgument.java to make a program UseThree.java that takes three names and prints out a proper sentence with the names in the reverse of the order given,

so that for example, "java UseThree Alice Bob Carol" gives "Hi Carol, Bob, and Alice.".

Exercise

Write a program TenHelloWorlds.java that prints "Hello, World" n times, where n is input by the user.

Exercise – What is the output?

```
public class Hello {
    public static void main() {
        System.out.println("Doesn't execute");
    }
}
```

Elements of the Java programming language

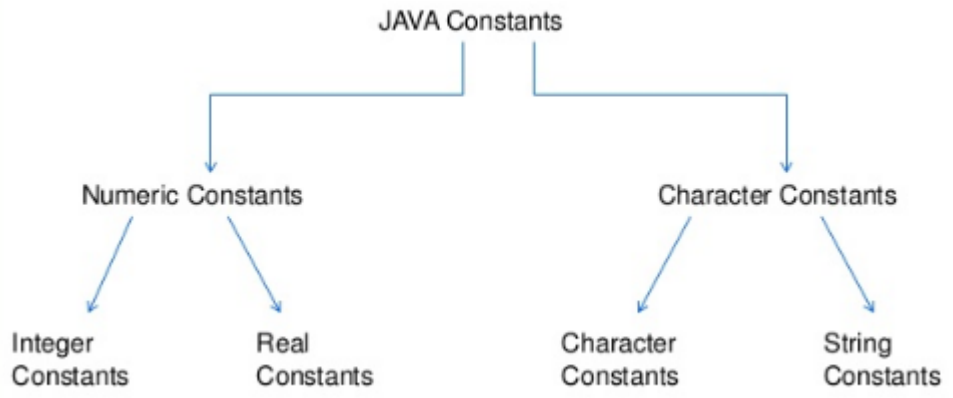
MindsMapped Consulting

JAVA CONTROLS

- Variables and Constants
- Arithmetic Operator and Expressions
- Type Conversion in Java
- Java's Control Statements
  - If
  - If-else
  - Do-while
  - While
  - for
  - Increment and Decrement Operators
  - Escape Sequences Characters
  - Relational and Logical Operators
  - Ternary Operators
  - Switch case
  - Break
  - Bitwise Operators
  - Arrays-Single and Multidimensional

Variables and Constants in Java

Constants: Constant "literals" in Java refer to fixed values that do not change during the execution of a program.
Java supports several types of constants given in figure below :



Integer Constants: Refers to a sequence of digits. There are three types of Integers, namely, decimal, octal and hexadecimal integer.

Decimal Integer consist of a set of digits, 0 through 9, preceded by an optional minus sign.

An octal integer constant consists of any combination of digits from the set 0 through 7, with a leading 0.

A sequence of digits preceded by ox or OX is considered as hexadecimal integer. They may also include alphabets A through F.

Variables and Constants

Real Constants: Integer constant are inadequate to represent quantities that vary continuously, such as distance, heights, temperature, prices and so on. These quantities are represented by numbers containing fractional parts like 17.546. Such numbers are called real.

The real number may also be expressed in exponential (or scientific) notation. For example, the value 215.65 may be written as 2.1565e2 in exponential notation. e2 means multiply by 10^2. The general form is :

mantissa e exponent

mantissa is either a real number expressed in decimal notation or an integer. The exponent is an integer with an optional plus or minus sign. The letter e separating the mantissa and the exponent can be written in either lowercase or uppercase. Since the exponent causes the decimal point to "float", this notation is said to represent a real number in floating point form.

Variables and Constants in Java

Single Character Constants: A single character constant (or simply character constant) contains a single character enclosed within a pair of single quote marks. Examples of character constants are : '5' 'X' ';'

String Constant: A string constant is a sequence of characters enclosed between double quotes. The characters may be alphabets, digits, special characters and blank spaces. Example are : "Hello Java" "1997"

**** Constants in Java are not directly supported. Then how do you declare constants?***

Variable: A variable is an Identifier that denotes a storage location used to store a data value. Unlike constants that remain unchanged during the execution of program. Examples of variables : average,height,total_height.

Variable name may consist of alphabets, digits, the underscore(_) and dollar characters.

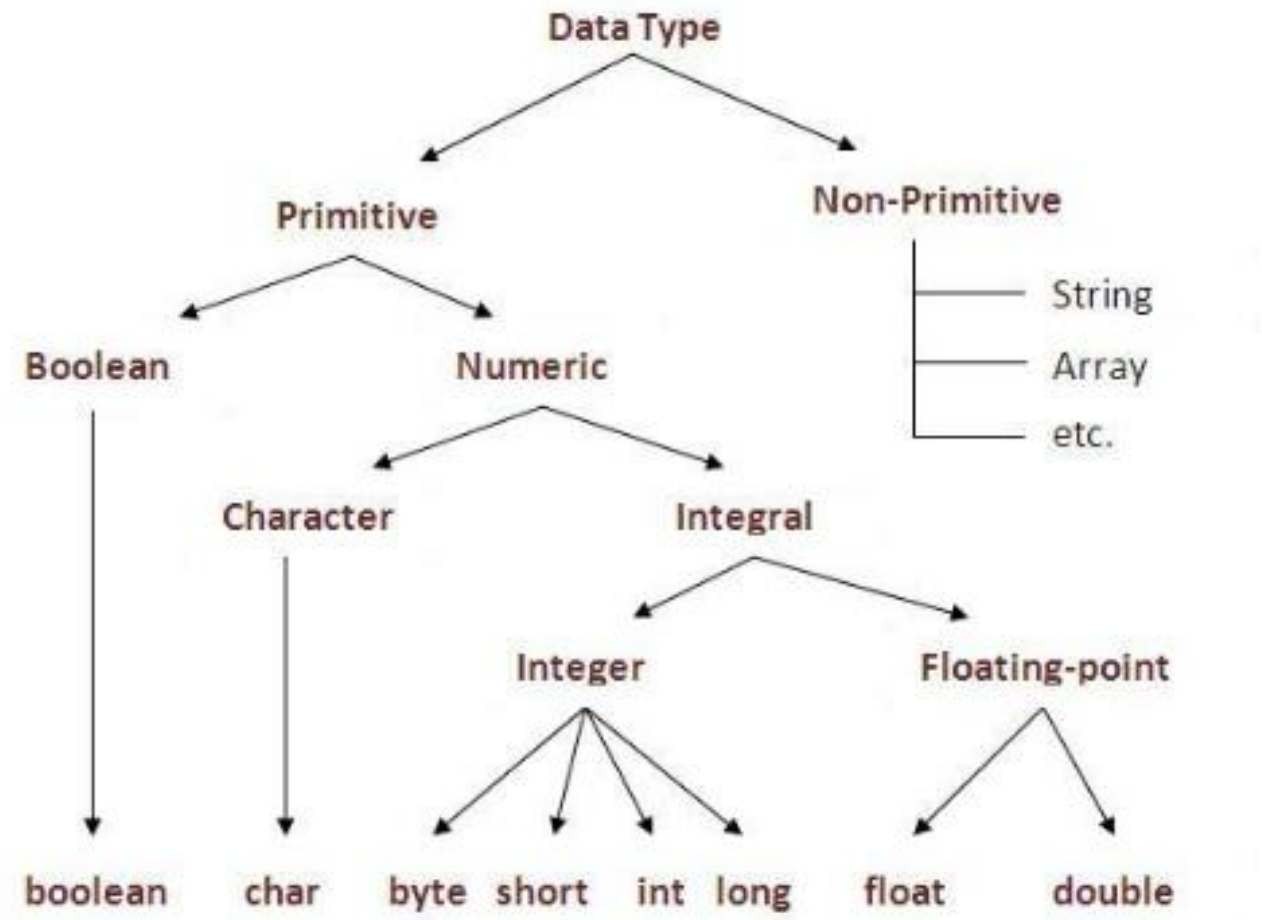
Rules to write Variable/Identifier in Java :

- They must not begin with digit
- Upper and lowercase are distinct. This means that the variable Total is not the same as total or TOTAL.
- It should not be a keyword.
- White space is not allowed.
- Variable names can be of any length.

Built-in Data Types

MindsMapped Consulting

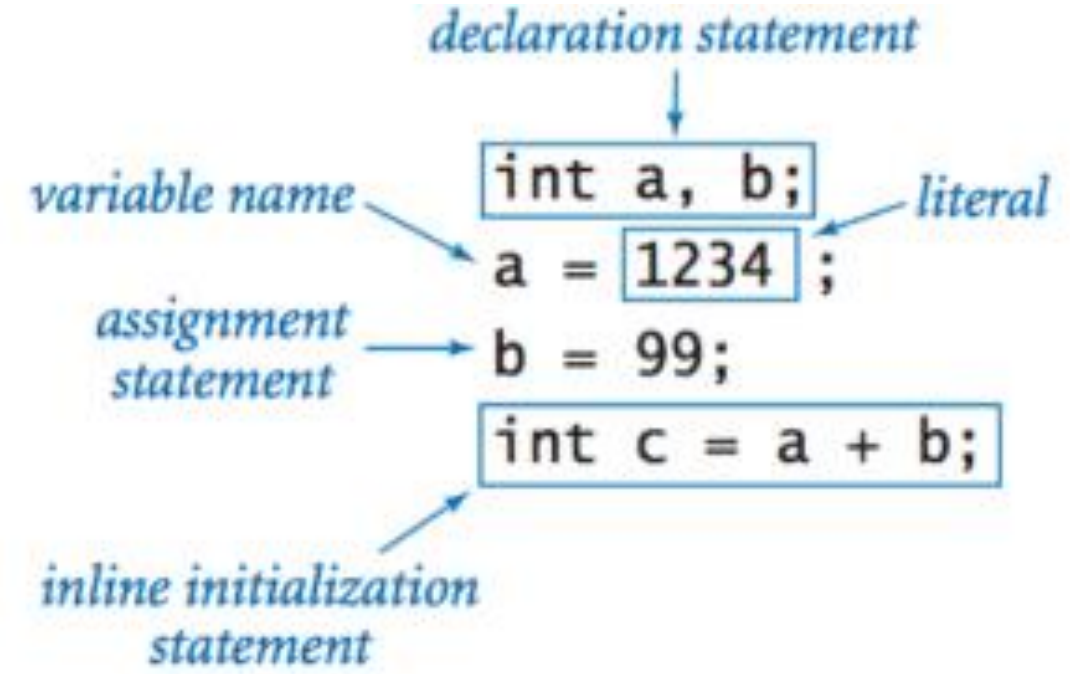
Data Types in Java



Commonly Used Built-in Data Types

type	*set of values*	*common operators*	*sample literal values*
`int`	integers	`+ - * / %`	`99 12 2147483647`
`double`	floating-point numbers	`+ - * /`	`3.14 2.5 6.022e23`
`boolean`	boolean values	`&& || !`	`true false`
`char`	characters		`'A' '1' '%' '\n'`
`String`	sequences of characters	`+`	`"AB" "Hello" "2.5"`

Declaration and Assignment Statements



Exercise: What is the output?

```
String ruler1 = " 1 ";
String ruler2 = ruler1 + "2" + ruler1;
String ruler3 = ruler2 + "3" + ruler2;
String ruler4 = ruler3 + "4" + ruler3;
String ruler5 = ruler4 + "5" + ruler4;

System.out.println(ruler1);
System.out.println(ruler2);
System.out.println(ruler3);
System.out.println(ruler4);
System.out.println(ruler5);
```

Integers - Exercise

expression	*value*	*comment*
`99`	99	*integer literal*
`+99`	99	*positive sign*
`-99`	-99	*negative sign*
`5 + 3`	8	*addition*
`5 - 3`	2	*subtraction*
`5 * 3`	15	*multiplication*
`5 / 3`	1	*no fractional part*
`5 % 3`	2	*remainder*
`1 / 0`		*run-time error*
`3 * 5 - 2`	13	** has precedence*
`3 + 5 / 2`	5	*/ has precedence*
`3 - 5 - 2`	-4	*left associative*
`( 3 - 5 ) - 2`	-4	*better style*
`3 - ( 5 - 2 )`	0	*unambiguous*

Integers

values	integers between -2^{31} and $+2^{31}-1$					
typical literals			1234 99 0 1000000			
operations	*sign*	*add*	*subtract*	*multiply*	*divide*	*remainder*
operators	+ -	+	-	*	/	%

Integers – Exercise – Try these…

expression	*value*	*comment*
`99`	99	*integer literal*
`+99`	99	*positive sign*
`-99`	-99	*negative sign*
`5 + 3`	8	*addition*
`5 - 3`	2	*subtraction*
`5 * 3`	15	*multiplication*
`5 / 3`	1	*no fractional part*
`5 % 3`	2	*remainder*
`1 / 0`		*run-time error*
`3 * 5 - 2`	13	** has precedence*
`3 + 5 / 2`	5	*/ has precedence*
`3 - 5 - 2`	-4	*left associative*
`( 3 - 5 ) - 2`	-4	*better style*
`3 - ( 5 - 2 )`	0	*unambiguous*

Floating Point Numbers: double

values	real numbers (specified by IEEE 754 standard)			
typical literals	`3.14159`	`6.022e23`	`2.0`	`1.4142135623730951`
operations	*add*	*subtract*	*multiply*	*divide*
operators	+	-	*	/

Floating Point Numbers: Exercise – Try these..

expression	*value*
`3.141 + 2.0`	`5.141`
`3.141 - 2.0`	`1.111`
`3.141 / 2.0`	`1.5705`
`5.0 / 3.0`	`1.6666666666666667`
`10.0 % 3.141`	`0.577`
`1.0 / 0.0`	`Infinity`
`Math.sqrt(2.0)`	`1.4142135623730951`
`Math.sqrt(-1.0)`	`NaN`

Booleans

values	*true or false*		
literals	`true`	`false`	
operations	and	or	not
operators	&&	||	!

a	*!a*
`true`	`false`
`false`	`true`

a	*b*	*a && b*	*a || b*
`false`	`false`	`false`	`false`
`false`	`true`	`false`	`true`
`true`	`false`	`false`	`true`
`true`	`true`	`true`	`true`

Comparison Operators

op	*meaning*	*true*	*false*
==	*equal*	2 == 2	2 == 3
!=	*not equal*	3 != 2	2 != 2
<	*less than*	2 < 13	2 < 2
<=	*less than or equal*	2 <= 2	3 <= 2
>	*greater than*	13 > 2	2 > 13
>=	*greater than or equal*	3 >= 2	2 >= 3

Exercise – Leap Year

Write a program to check if provided year is leap year or not.

Parsing Command Line Arguments (Converting Strings to Primitives)

`int`	`Integer.parseInt(String s)`	*convert* `s` *to an* `int` *value*
`double`	`Double.parseDouble(String s)`	*convert* `s` *to a* `double` *value*
`long`	`Long.parseLong(String s)`	*convert* `s` *to a* `long` *value*

Type Conversion

We often find ourselves converting data from one type to another using one of the following approaches.

- **Explicit type conversion**. Call methods such as Math.round(), Integer.parseInt(), and Double.parseDouble().

- **Automatic type conversion**. For primitive numeric types, the system automatically performs type conversion when we use a value whose type has a larger range of values than expected.

- **Explicit casts**. Java also has some built-in type conversion methods for primitive types that you can use when you are aware that you might lose information, but you have to make your intention using something called a cast.

- **Automatic conversions for strings.** The built-in type String obeys special rules. One of these special rules is that you can easily convert any type of data to a String by using the + operator.

Exercise: Evaluate the following

```
(1 + 2 + 3 + 4) / 4.0
Math.sqrt(4)
"1234" + 99
11 * 0.25
(int) 11 * 0.25
11 * (int) 0.25
(int) (11 * 0.25)
(int) 2.71828
Math.round(2.71828)
(int) Math.round(2.71828)
Integer.parseInt("1234")
```

Exercise: Evaluate the following - Answers

expression	*expression type*	*expression value*
`(1 + 2 + 3 + 4) / 4.0`	`double`	`2.5`
`Math.sqrt(4)`	`double`	`2.0`
`"1234" + 99`	`String`	`"123499"`
`11 * 0.25`	`double`	`2.75`
`(int) 11 * 0.25`	`double`	`2.75`
`11 * (int) 0.25`	`int`	`0`
`(int) (11 * 0.25)`	`int`	`2`
`(int) 2.71828`	`int`	`2`
`Math.round(2.71828)`	`long`	`3`
`(int) Math.round(2.71828)`	`int`	`3`
`Integer.parseInt("1234")`	`int`	`1234`

Mathematical Functions

public class Math	
double abs(double a)	*absolute value of a*
double max(double a, double b)	*maximum of a and b*
double min(double a, double b)	*minimum of a and b*
double sin(double theta)	*sine of theta*
double cos(double theta)	*cosine of theta*
double tan(double theta)	*tangent of theta*
double toRadians(double degrees)	*convert angle from degrees to radians*
double toDegrees(double radians)	*convert angle from radians to degrees*
double exp(double a)	*exponential* (e^a)
double log(double a)	*natural log* ($\log_e a$, *or* $\ln a$)
double pow(double a, double b)	*raise a to the bth power* (a^b)
long round(double a)	*round a to the nearest integer*
double random()	*random number in* [0, 1)
double sqrt(double a)	*square root of a*
double E	*value of e (constant)*
double PI	*value of* π *(constant)*

Mathematical Functions: Examples

method call	*library*	*return type*	*value*
`Integer.parseInt("123")`	`Integer`	`int`	`123`
`Double.parseDouble("1.5")`	`Double`	`double`	`1.5`
`Math.sqrt(5.0*5.0 - 4.0*4.0)`	`Math`	`double`	`3.0`
`Math.log(Math.E)`	`Math`	`double`	`1.0`
`Math.random()`	`Math`	`double`	*random in* [0, 1)
`Math.round(3.14159)`	`Math`	`long`	`3`
`Math.max(1.0, 9.0)`	`Math`	`double`	`9.0`

Exercise: Random Integer

Read an integer command-line argument *n* and print a "random" integer between 0 and *n*−1.

Hint: Use Math.random()

Exercise: Random Integer

Read an integer command-line argument *n* and print a "random" integer between 0 and *n*−1.

Hint: Use Math.random()

```
public class RandomInt {
    public static void main(String[] args) {
        // a positive integer
        int n = Integer.parseInt(args[0]);

        // a pseudo-random real between 0.0 and 1.0
        double r = Math.random();

        // a pseudo-random integer between 0 and n-1
        int value = (int) (r * n);

        System.out.println(value);
    }
}
```

Exercise

A physics student gets unexpected results when using the code

```
double force = G * mass1 * mass2 / r * r;
```

to compute values according to the formula $F = Gm_1m_2 / r^2$. Explain the problem and correct the code.

Exercise: What is the output?

```
boolean isA = (90 <= grade <= 100);
```

```
double x = (double) (3/5);
System.out.println(x);
```

```
int x = 65536;
long y = x * x;
System.out.println(y);
```

```
(Math.sqrt(2) * Math.sqrt(2) == 2)
```

Summary of Operators

Simple Assignment Operator

=	Simple assignment operator

Arithmetic Operators

+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

Summary of Operators

Unary Operators

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Summary of Operators

Equality and Relational Operators

==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Conditional Operators

&&	Conditional-AND
||	Conditional-OR
?:	Ternary (shorthand for if-then-else statement)

Summary of Operators

Type Comparison Operator

`instanceof`	Compares an object to a specified type

Bitwise and Bit Shift Operators

`~`	Unary bitwise complement
`<<`	Signed left shift
`>>`	Signed right shift
`>>>`	Unsigned right shift
`&`	Bitwise AND
`^`	Bitwise exclusive OR
`|`	Bitwise inclusive OR

Exercise – What is the output?

In the following program, explain why the value "6" is printed twice in a row:

```
class PrePostDemo {
    public static void main(String[] args){
        int i = 3;
        i++;
        System.out.println(i);    // "4"
        ++i;
        System.out.println(i);    // "5"
        System.out.println(++i);  // "6"
        System.out.println(i++);  // "6"
        System.out.println(i);    // "7"
    }
}
```

What is the output?

Consider the following code snippet.

```
int i = 10;
int n = i++%5;
```

- What are the values of i and n after the code is executed?
- What are the final values of i and n if instead of using the postfix increment operator (i++), you use the prefix version (++i))?

Write unambiguous expressions…

```
x + y / 100     // ambiguous

x + (y / 100) // unambiguous, recommended
```

Write unambiguous expressions…

```
x + y / 100     // ambiguous

x + (y / 100) // unambiguous, recommended
```

Importance of Blocks in Code

A block is a group of zero or more statements between balanced braces and can be used anywhere a single statement is allowed. The following example, BlockDemo, illustrates the use of blocks:

```
class BlockDemo {
     public static void main(String[] args) {
          boolean condition = true;
          if (condition) { // begin block 1
               System.out.println("Condition is true.");
          } // end block one
          else { // begin block 2
               System.out.println("Condition is false.");
          } // end block 2
     }
}
```