ES1036: Programming Fundamentals

Unit 2: Java Basics

"There are no secrets to success. It is the result of preparation, hard work, and learning from failure" ~Colin Powell

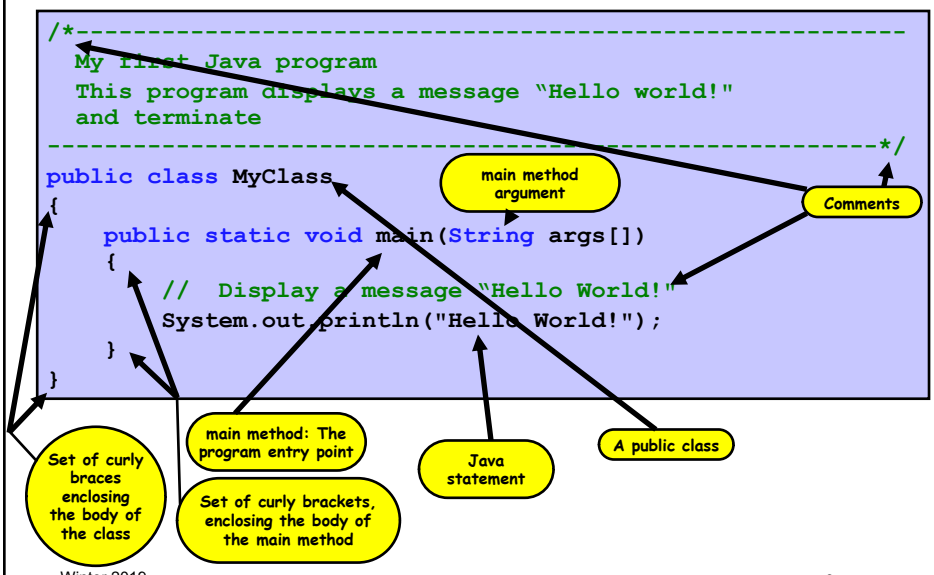
"Genius is 1% talent and 99% percent hard work..." ~ Albert Einstein

Dr. Quazi M. Rahman, Ph.D, P.Eng., SMIEEE
Office Location: TEB 361
Email: QRAHMAN@eng.uwo.ca
Phone: 519-661-2111 x81399

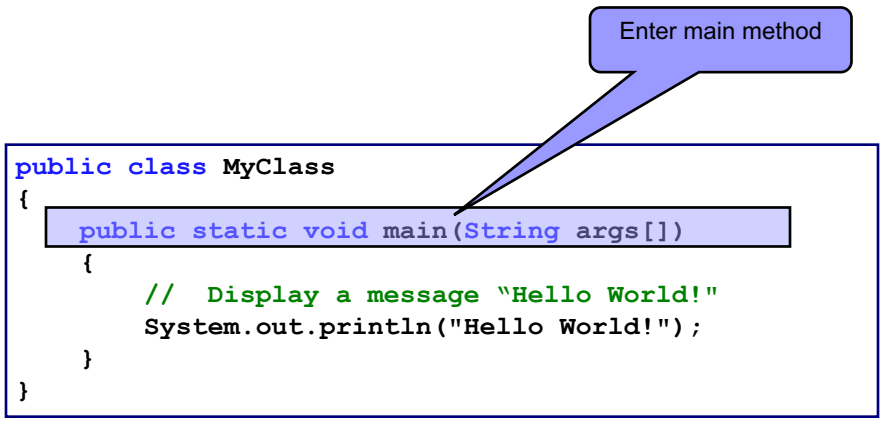
Outline

- Program Structure
- Constants and Variables
- Java Operators
- Standard Input and Output
- Basics Math Functions
- Arithmetic conversions
- Programming Style and Documentation (Reading assignment)
- Displaying formatted output (For lab only)

Program Structure

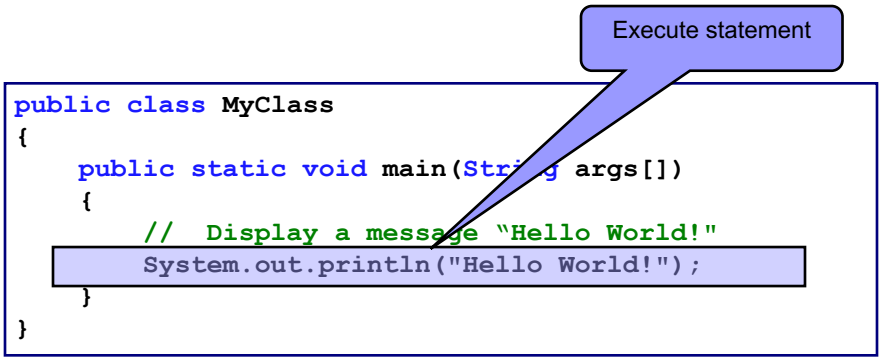


Trace The Program Execution



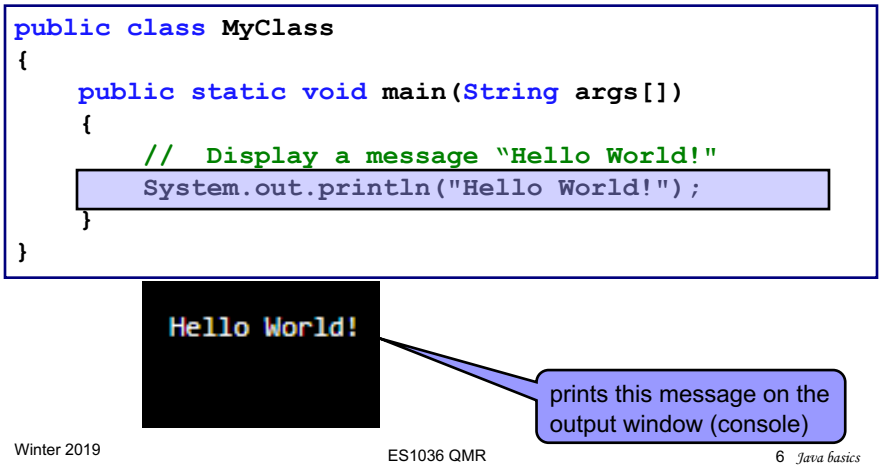
Reference:
https://docs.oracle.com/javase/10/docs/api/java/lang/System.html

Trace The Program Execution



Note: (http://docs.oracle.com/javase/10/docs/api/java/lang/System.html)
System is a class; out is a PrintStream (another class) type static object declared inside System class (System class's field); println() is a method defined inside the PrintStream class.

Trace The Program Execution



Anatomy of a Java Program

- Class Name
- The main method
- Comments
- Package
- Reserved words
- Modifiers
- Blocks
- Statements
- println()/print() method

Class Name

Every Java program must have at least one class. Each class has a name. By convention, class names start with an uppercase letter. In the following example, the class name is MyClass. Any valid name (discussed later) can be chosen.

```
/*------------------------------------------------------------
  My first Java program
  This program displays a message "Hello world!"
  and terminate
------------------------------------------------------------*/
public class MyClass
{
    public static void main(String args[])
    {
        //  Display a message "Hello World!"
        System.out.println("Hello World!");
    }
}
```

What is Class and object? (Discussed later)

- A Class is a template, blueprint or contract that defines the data and behavior associated with the instances of that **class**.
- An object is a instance of a class. When you instantiate a **class** you create an object that looks and feels like other instances of the same **class**.
- E.g. 'Student' is a class; The students Jess, Mike are the instances of the Student Class.

The `main()` method

- It is the entry point of any Java program. Every Java program starts executing its instructions from the *main* method.
- *A method (a function) is a collection of statements that are grouped together to perform an operation.*
- The main method-header has the form:

```
public static void main(String[] args) or
public static void main(String[] anyName)
```

- **`anyName`** has to be a Java-accepted **valid name** (discussed later).

```
public static void main(String x[]) {
    System.out.println("Hello World!");
}
```

Note: Java source programs are case sensitive. It would be wrong, (for example) to replace **`main`** in the program with **`Main`**

`main()` method *continues*

- All statements in the main method (and in all other methods) must be enclosed in a statement-block that starts with left curly brace '`{`' and ends with right curly brace '`}`'
- A java application cannot run unless one of the classes has a main() method.
- The JVM loads a class and starts executing different instructions by calling the `main(String[] args)` method
  - `public`: the method can be called by any object (Discussed later)
  - `static`: no object needs to be created first [JVM does not create any object before the program runs. So, to run the program main method should be invoked/called without creating any object. That's why main should be static] (Discussed later)
  - `void`: nothing will be returned from this method (Discussed later)

Comments in Java

- Comments allow programmers to put some descriptions inside the code (For this course, writing comments is mandatory for all)
- Comments are not programming statements and thus are ignored by the complier
- Two types: (check the code on the previous slide)
  - *Line comment*: is preceded by two slashes `//` on a line
  - *Paragraph comment*: is enclosed between `/*` and `*/` on one or several lines
- When the compiler sees `//`, it ignores all text after `//` on the same line
- When it sees `/*`, it looks for the next `*/` and ignores any text between `/*` and `*/`

Comments in Java continues (FYI)

- *javadoc comments*:
  - javadoc comments begin with /** and end with */.
  - They are used for documenting classes, data, and methods.
  - They can be extracted into an HTML file using JDK's javadoc command.

Package

■A **Java package** (keyword `package`) is a mechanism for organizing Java classes into a file whose name can be an existing one or a created one (`package myPackage1;`)

■ Java packages can be stored in compressed files called JAR (Java archive) files, allowing classes to download faster as a group rather than one at a time.

■Programmers also typically use packages to organize classes belonging to the same category or providing similar functionality.

Note: Packages in Java

- Every class in Java belongs to a package.
- The class is added to the package when it is compiled.
- To put a class in a specific package, you need to add the following line as the first non-comment and nonblank statement in the program:

```
package packagename;
```

For example:

```
package ca.uwo.mypackage;
```


Using Classes from Packages

- There are two ways to use classes from a package.
- One way is to use the fully qualified name of the class.
  - For example, the fully qualified name for `JOptionPane` is `javax.swing.JOptionPane`.
- The other way is to use the import statement.
  - For example, to import all the classes in the javax.swing package, use `import javax.swing.*;` (*Note: wild card notation can only be used for classes, NOT for packages*)
- You can also import a specific class.
  - For example, this statement imports javax.swing.JOptionPane:

```
import javax.swing.JOptionPane;
```

- The information for the classes in an imported package is not read in at compile time or runtime unless the class is used in the program. The import statement simply tells the compiler where to locate the classes.

Core Java Packages (Under the API Library)

- java.lang
  - Provides classes that are fundamental to the design of the Java programming language
    - Includes wrapper classes (Integer, Double etc), String and StringBuffer, Object, StringBuilder, Math…
    - Imported implicitly into all packages.
- java.util
  - Contains the collections framework, event model, date and time facilities, internationalization, and miscellaneous utility classes
    - Date, Scanner, Arrays…
- java.io
  - Provides for system input and output through data streams, serialization and the file system.
    - PrintWriter, BufferReader, IOException, InputStreamReader etc.

- **java.math**
  - Provides classes for performing arbitrary-precision integer arithmetic (BigInteger) and arbitrary-precision decimal arithmetic (BigDecimal)
- **java.sql**
  - Provides the API for accessing and processing data stored in a data source (usually a relational database). E.g., Time
- **java.text**
  - Provides classes and interfaces for handling text, dates, numbers, and messages in a manner independent of natural languages

http://docs.oracle.com/javase/10/docs/api/overview-summary.html
https://docs.oracle.com/javase/tutorial/java/concepts/package.html

Reserved Words / Keywords

- Reserved words or keywords are the words that have a specific meaning to the compiler and cannot be used for other purposes in the program.
- If any keyword is used as an identifier (name of a variable, class etc.), an error or unexpected result will occur.
- Let's play! In the example below, what are the reserved words?

```
public class MyClass
{
    public static void main(String args[])
    {
        System.out.println("Hello World!");
    }
}
```

Java Keywords (FYI)

Note: Most Integrated Development Environments (IDEs) for Java, use syntax highlighting to display keywords in a different colour for easy identification.

abstract	do	if	private	this
assert	double	implements	protected	throw
boolean	else	import	public	throws
break	enum	instanceof	return	transient
byte	extends	int	short	true
case	false	interface	static	try
catch	final	long	strictfp	void
char	finally	native	super	volatile
class	float	new	switch	while
const	for	null	synchronized	continue
default	goto	package		

https://en.wikipedia.org/wiki/List_of_Java_keywords

Modifiers

- Java uses certain reserved words called modifiers that specify the properties of the data members (field), methods, and classes and how they can be used.
- Examples of modifiers are (discussed later)
  - public
  - static
  - private
  - final
  - abstract
  - protected
- Only final modifier can be used for any local variable or local object reference (discussed later) in Java.

Note: Declaring Array Variables in Java (Discussed later)

- `datatype[] arrayRefVar;`

  Example:

  ```
  ..... main(String[] args) {……}

  double[] myList;
  ```

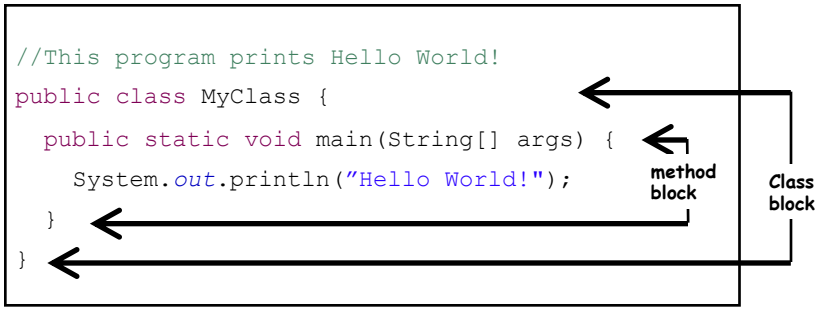
- `datatype arrayRefVar[]; // This style is allowed, but not preferred because arrayRefVar is not an array, it's an array reference`

  Example:

  ```
  double myList[];
  ```


Blocks

- A pair of curly braces in a program forms a block that groups components of a program.



Java Statements

- A statement in Java is an instruction written in Java-Syntax that ends with a semicolon (might take one or more lines).
- Different types of statement include:
  - Declaration statements
  - Assignment statements
  - Compound statements (also known as block statements)
    - A group of JAVA statements (One or more) enclosed between curly braces '{' and '}', is known as compound/block statement.
  - Selection statements

println() Method

- println() method (defined in the PrintStream class) displays a message and/or data to the standard output (console or monitor)
- println() method has the general form as:
  - `System.out.println(expression+expression+…);`
- Note: An expression can be any Java expression such as, string constant (*one or more characters enclosed by double-quotation marks*), character constant (*single character enclosed by single-quotation marks*), identifier (*variable name*), formula (*arithmetic and/or logical expressions*) or method call. Two or more different **types** of expressions are concatenated by the '+' sign.
- Example:

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

This statement outputs the string, ***Hello world!***, to the console and then sends the cursor to the next line (println→ print something and then go to the next line)

Note: A string-constant must be enclosed in double quotation marks. A character-constant must be enclosed in single quotation marks.

Example code with *println()* method

```
public class MyClass {
  public static void main(String args[]) {
  /*variable declaration and initialization*/
  int x = 10, y = 23;
  //string constant
  System.out.println("Do nothing");
  //character constant
  System.out.println('?');
  //identifier
  System.out.println("x = " + x);
  /*equation or formula*/
  System.out.println(x + y - 3);
  //method call
  System.out.println(Math.sqrt(x));
  //numerical constant
  System.out.println(12.4);
  }
}
```


`print()` method: A Variation of `println()` method

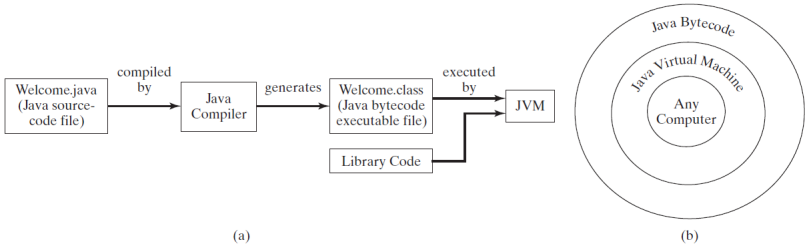
- `print()` method has the general form as:
  - `System.out.println(expression+expression+…);`
- It has the same functionality as println() method except for the fact that it does not generate a new line character after printing the required information on the console.

Let's guess the output:

```
public class MyClass {
  public static void main(String args[]) {
  /*variable declaration and initialization*/
  int x = 10, y = 23;
  //string constant
  System.out.print("The answer is: ");
  //character constant
  System.out.println("YES! ");
  }
}
```

Converting Java programs in to machine language: The steps

- Text Editor
- Compiler
- Bytecode (Virtual Machine Code)
- Java Virtual Machine (JVM)



http://www.cs.cmu.edu/~jcarroll/15-100-s05/supps/basics/history.html

Converting Java program in to machine language: Text Editor

- Text editor is JAVA-text editing software.
- The program (the set of instructions), known as source code, is written using the text editor.
- The source code (human readable instructions) is saved on to the secondary storage (External memory) with an extension '.java' to let the compiler know that it is written in Java language.
- The source code needs to be grammatically (Java grammar) correct.
- If the name (your pick) of the project is Welcome, the source code is saved as Welcome.java.

Converting Java in to machine language: Compiler

- A Compiler translates the source code into machine readable instructions, provided that the source code is grammatically correct.
- The compiler in Java (called `javac`) translates the source code into virtual machine code (known as bytecode), with an extension **.class**.
- If the name of the source file is Welcome.java, then the name of the bytecode will be Welcome.class.

Converting Java in to machine language: Bytecode

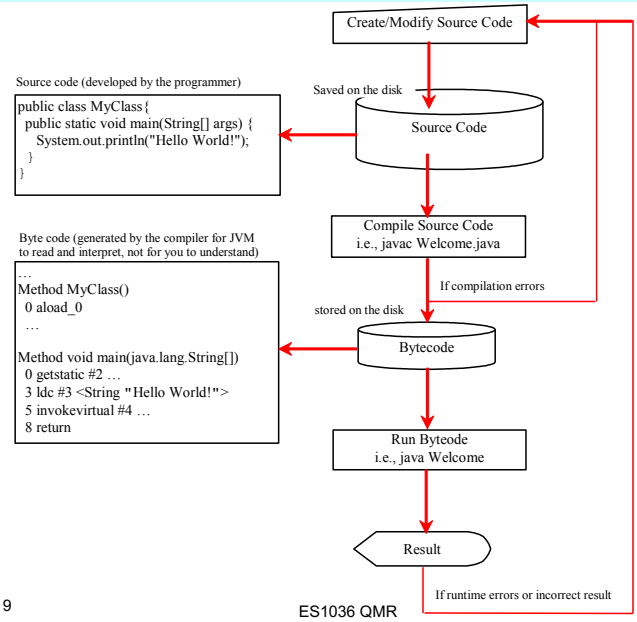
- Bytecode is a Virtual Machine code for Java
- The bytecode is machine-independent and can run on any machine that has a Java interpreter (Name of the Java interpreter: java).
- Java interpreter is a part of the Java Virtual Machine (JVM).
- A virtual machine (VM) is a software implementation of a machine (computer) that executes programs like a real machine.

Converting Java in to machine language: JVM

- JVM is a part of Java Runtime Environment (JRE).
  - JRE is a platform-specific environment.
  - One has to make sure that the proper version of the JRE is installed in the end-user system.
- JVM executes the .class file on the operating system that it sits on.
  - At runtime the Byte Code is loaded, checked and run by the Java interpreter which is a part of JVM.

https://www.geeksforgeeks.org/differences-jdk-jre-jvm/

Creating, Compiling, and Running Programs



FYI: Converting Java in to machine language with the aid of an IDE

- Integrated Development Environment (IDE) program: An IDE is a software application that provides comprehensive facilities to computer programmers for software development.
- An IDE normally consists of a source code editor, build automation tools, and a debugger.
- Most modern IDEs have intelligent code completion.
- Some IDEs, such as NetBeans and Eclipse, contain a compiler, interpreter, or both.
  - Debugger: The debugger locates the problem in the program during the compilation time (more information on debugger will be available on the OWL home page soon)

Converting Java in to machine language: The errors

- Syntax errors: (also called parse errors, compilation error) reported by the compiler once any syntactical error is made. The program will not compile with erroneous syntax.
- Runtime errors (Execution error) occur when the programming environment detects an operation that is impossible to carry out during program execution. Runtime errors are generally encountered due to memory mismanagement issue.
- Logical errors (or program bugs) occur when a program doesn't perform the way it is expected to perform.

Review

1) Computer can execute the code in _____.

a) machine language
b) assembly language
c) high-level language
d) none of the above

Review

2) The extension of a Java virtual object file is

a) .java
b) .obj
c) .class
d) .exe

Review

3) The extension of a Java source code file is

a) .java
b) .obj
c) .class
d) .exe
e) .cpp

Review

4) True or False: The source code in Java is compiled into a bytecode.

a) True
b) False

Review

5) True or False: Bytecode is machine dependent.

a) True
b) False

Review

6) True or False: In Java programming language, the main() method has to be declared as a static method of a class.

a) True
b) False

Review

7) In the following java statement what does `System` represent?

```
System.out.println("Welcome to Java!");
```

a) A package
b) A class
c) An object
d) A method
e) None of the above

Java Variables (Objects)

- In the programming environment we need to *reserve memory spaces* to process any data or information
- These reservations are done using variable (and/or constant) names
- Variables are boxes or placeholders in the memory that can hold things
- Each variable (box) has to be identified by a name, which has to be declared before using it. The name of any type of variable (or constant, function, namespace, class etc) is called an "identifier"
- Size of the variable (box) depends on the "type" of things we are planning to store there
- We have to tell the compiler in advance ("declare"),
  - Names of each of the variables (boxes) we want
  - The type of things that will be put in each variable (box)

Identifier characteristics

- An identifier (name of a variable, constant, function, class, struct, namespace etc) may consist of
  - Any letters (A to Z, both capital and small),
  - Any digits (0 to 9), and
  - Two special characters : underscore (_) and dollar ($).
- An identifier cannot start with a digit
- An identifier cannot be a reserved word
- An identifier can be of any length, but your compiler may impose some restriction. Use identifiers of 31 characters or fewer to ensure portability

For example: **`radius, A, anyName, _number, surfaceArea`** are legal identifiers, but **`2A, yes&No,`** and **`d+4`** are not

Naming Conventions

- Choose meaningful and descriptive names. To store the radius value, choose a name `radius` instead of `x` or `y`.
  - Note: If you are calculating an expression such as: $y = x^2 + 3z$, then it will be a good idea to choose the same variable names `x`, `y` and `z` in the program.
- Variables and method (function) names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
    - For example, the variables `interestRate` and `year`, and the function `computePayment`.

Naming Conventions, *cont.*

- Class names (*Class will be discussed later*):
  - Capitalize the first letter of each word in the name. For example, the class name `InterestRateCalculation`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `MAX_VALUE` or `AGE`.

Variable Declaration

- To use a variable, one has to declare it by writing the name of the variable as well as the data type it represents. This is called variable declaration
- Declaring a variable tells the compiler to allocate enough memory space to hold a value of this data type and to associate the identifier with this location
- Here is the syntax (grammar) for declaring a variable:

```
datatype variableName;
```

Example of declaration statements:

```
int x;              //Declare x to be an integer variable
double radius;      //Declare radius to be a double variable
double surfaceArea; //Declare surfaceArea to be a double variable
char a;             //Declare a to be a character variable
bool xy;             //Declare xy to be a boolean variable
```

- *Note: It's a good practice to pick up a meaningful name for a variable. Above: radius and surfaceArea are good names.*

Some built-in (primitive) Data Types

- *IEEE 754 floating point numbering system:* http://en.wikipedia.org/wiki/IEEE_floating_point

Name	Range	Storage Size
byte	-2^7 (-128) to 2^7-1 (127)	8-bit signed int.
short	-2^{15} (-32768) to $2^{15}-1$ (32767)	16-bit signed int.
int	-2^{31} (-2147483648) to $2^{31}-1$ (2147483647)	32-bit signed int.
long	-2^{63} to $2^{63}-1$ (i.e., -9223372036854775808 to 9223372036854775807)	64-bit signed int.
float	Negative range: -3.4028235E+38 to -1.4E-45 Positive range: 1.4E-45 to 3.4028235E+38	32-bit IEEE 754
double	Negative range: -1.7976931348623157E+308 to -4.9E-324 Positive range: 4.9E-324 to 1.7976931348623157E+308	64-bit IEEE 754

Assigning a Value to a Variable

- After a variable is declared, one can assign a value to it; this is called variable initialization.
- It's a good and safe practice to initialize a variable (or object) when it is declared
- *Note: If any uninitialized variable is used in any expression, compilation error will be generated.* In Class discussion with example?
- Variable value can also be assigned using an assignment statement
- The syntax for assignment statement is:

```
variable = expression;
```

- An *expression* represents a computation involving
  - values,
  - variables, and
  - operators that altogether evaluates to a value
- The equal sign (=) is known as the assignment operator

Examples on variable declaration and initialization

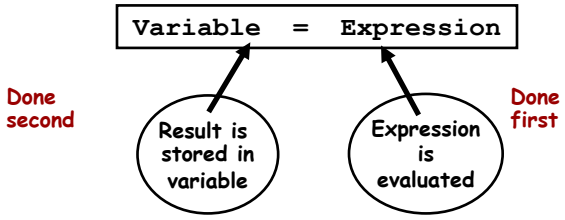
```
// declare x as an integer variable and assign 1 to it
int x =1; //method 1: declaration and initialization
// declare radius as a double type data variable and assign 1 to it
double radius = 1.0; //method 1
double radius = 1; //method 2
/*each of the above statements can also be written using two statements as follows:*/
double radius; //variable declaration
radius =  1; // assigning a value to a declared variable
//declare a variable called SurfaceArea
double surfaceArea;
//compute surface area by using the declared variable radius
surfaceArea = 4 * radius * radius * 3.14159; //an assignment statement
//Declare a to be a character variable
char a;
// assign the letter K to the character variable a
a = 'K';
```


Literals

- A *literal* is a constant value that appears directly in the program.
- This is also referred as *hard-coding*
- Examples: in the following statements, 34, 1,000,000, 5.5 and 'A' are literals:

```
int i = 34;
long x = 1000000;
double d = 5.5;
char k = 'A';
```

Assignment Expression



- A variable can appear in both sides of the assignment operator but left hand side has to be a writable memory location (*In-Class discussion*).
- For example: `int x = 10; x = x + 1;`
  - First the right hand side term `x + 1` is evaluated and then the result is assigned to the variable `x` on the left hand side
- In the example above, what will be the value of x after the second statement?

Review: Java Statements

- Examples:
  - Declaration statements
  - Assignment statements (no data type is labeled, and assignment operator '=' is used)

```
int sum = 0, a = 5, b =
 10;/*declaration and
  initialization*/
sum = a + b;

float temp_c =0;
float temp_f = 78;
temp_c = (temp_f -
 32)*5/9;
```

Where to Declare a variable?

- Immediately prior to use

```
public static void main(String[] arg){
   System.out.println ("Adding two numbers:");
   int sum =0, a=4, b=5;
   sum = a + b;
   System.out.println (sum);
}
```

- At the beginning

```
public static int main(String[] arg){
   int sum = 0, a=4, b=5;
   System.out.println ("Adding two numbers:");
   sum = a + b;
   System.out.println (sum);
}
```


Review

8) Name of a variable (or object) is used to:

a) Tell the computer how much memory this variable needs
b) Distinguish one variable from another
c) Identify the type of things that can be written to it
d) Both (a) and (b)

Review

9) Type of a variable (or object) is used to:

a) Tell the computer how much memory this variable needs
b) Identify the type of things that can be written to it
c) Distinguish one variable from another
d) Both (a) and (b)

Review

10) What type of a statement is

```
float w_lb, w_kg;
```

a) A pre-processor directive
b) A selection statement
c) A declaration statement
d) An assignment statement

Review

11) What type of a statement is

```
w_kg = w_lb * 0.454;
```

a) A pre-processor directive
b) A selection statement
c) A declaration statement
d) An assignment statement

Variable re-declaration is a syntax-error in any block statement

- Review: a group of any number of statements enclosed by `{` and `}` is known as Block / compound statement:
- Example: In the following block compilation/parse/syntax/grammatical error will be generated from *line x*;

```
{
  int x = 10; x = x + 13;
  double x = 21.4; //line x
}
```

- *Exception: Two same variable names can be used in two different code-blocks without any error. This is related to the scope (discussed in unit 5) of a variable.*

Reading numbers from the standard input (keyboard)

- One of the ways in reading a number from the keyboard is to use Scanner object which is available in the java.util package.
- To be able to use a Scanner object, one has to import this package.
- The syntax to input an integer number:

```
Scanner input = new Scanner(System.in);
int anyNumber = input.nextInt();
```

Note: (http://docs.oracle.com/javase/10/docs/api/java/lang/System.html)
System is a class; in is an InputStream (another class) type static object declared inside System class (System class's field);. System.in represents the standard input (keyboard) as the input stream source. nextInt() is a method defined inside the Scanner class to read the integer value as a token.

Reading numbers from the standard input (keyboard) *cont..*

Method	Description
nextByte()	reads an integer of the **byte** type.
nextShort()	reads an integer of the **short** type.
nextInt()	reads an integer of the **int** type.
nextLong()	reads an integer of the **long** type.
nextFloat()	reads a number of the **float** type.
nextDouble()	reads a number of the **double** type.

Reading String or Character from the standard input (keyboard) using Scanner object

Method	Description
next()	reads a string without any space in the string. E.g., ES1036
nextLine()	reads a string with space(s) in the string. e.g., ES1036b section 001
next().charAt(0)	reads a non white-space character.

- White space characters: the characters that result from the space-bar-key, enter-key and tab-key.

Reading String or Character from the standard input (keyboard) using Scanner object *cont..*

Example Program-Segment for practice:

```
System.out.print("Enter a character value: ");
Scanner input = new Scanner(System.in);
char c = input.next().charAt(0);
System.out.println("You entered " + c);
System.out.print("Enter a string: ");
String s = input.next();
System.out.println("You entered " + s);
System.out.print("Enter another string: ");
String sw = input.nextLine();
System.out.println("You entered " + sw);
```

Named Constants: `final` modifier

Syntax: `final datatype CONSTANTNAME = VALUE;`

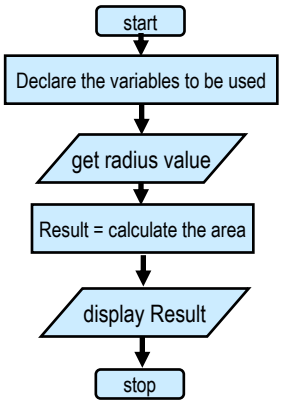
Examples:

```
final double PI = 3.14159; (PI has been
declared as a double type static final data
field in java.lang.Math class)

final int SIZE = 3;

final double E = 2.71828 (E, the base of
natural logarithm, has been declared as a
double type static final data field in
java.lang.Math class)
```


Example problem using constants and variables

- **Problem**
  Find the surface area ($4\pi r^2$) of a sphere. The radius (r) value of this sphere will be received from the user as an input.

Flow chart



The outline (we can call this 'algorithm')

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]) {
        // Step 1. Input the value of the radius of the sphere.

        // Step 2. Compute the surface area of the sphere.

        // Step 3. Display the values of the radius and
        //         the surface area of the sphere.

        // exit
  }
}
```

The Outline (more specific: approach 1)

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]) {
        /*Step 1. Input the value of the radius of the sphere.
        - declare a variable to store the radius
        - ask the user to input the value
        - input the value from the keyboard into the
          declared variable
        Step 2. Compute the surface area of the sphere.
        - declare a variable to store the area
        - compute the area and store the result into the
          declared variable
        Step 3. Display the values of the radius and
          the surface area of the sphere.

        Exit*/
    }
}
```


The Outline (more specific: approach 2)

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]) {
/*Prep: Variable and constant declarations
        - declare variables to store the radius and area values.
          Also, declare constant expression, if needed.
Step 1. Input the value of the radius of the sphere.
        - ask the user to input the value
        - input the value from the keyboard into the
          declared variable for radius
Step 2. Compute the surface area of the sphere.
        - compute the area and store the result into the
          declared variable for area
Step 3. Display the values of the radius and
         the surface area of the sphere.

Exit*/
  }
}
```

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]) {
    // Variable and constant declarations
    double radius, SurfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius of the sphere.
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere: ");
    radius = input.nextDouble();

    // Step 2. Compute the surface area of the sphere.
    SurfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.pow(radius,2);

    // Step 3. Display the values of the radius and
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + SurfaceArea + " sq cm\n";
 }
}
```



```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]){
    // Variable and constant declarations
    double radius, surfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius of the sphere.
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere: ");
    radius = input.nextDouble();

    // Step 2. Compute the surface area of the sphere.
    surfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.pow(radius,2);

    // Step 3. Display the values of the radius and
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + surfaceArea + " sq cm\n";
 }
}
```

Memory Diagram

Variable name	Value/Content
radius	?

Allocate memory for radius

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]){
    // Variable and constant declarations
    double radius, surfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius of the sphere.
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere: ");
    radius = input.nextDouble();

    // Step 2. Compute the surface area of the sphere
    surfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.pow(radius,2);

    // Step 3. Display the values of the radius and
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + surfaceArea + " sq cm\n";
 }
}
```

Memory Diagram

Variable name	Value/Content
radius	?
surfaceArea	?

Allocate memory for surfaceArea


```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]){
    // Variable and constant declarations
    double radius, surfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere: ");
    radius = input.nextDouble();

    // Step 2. Compute the surface area of the sphere
    surfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.p

    // Step 3. Display the values of the radius
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + surfaceArea + " sq cm\n";
 }
}
```

Memory Diagram

Variable name	Value/Content
radius	?
surfaceArea	?
PI	3.14159

Allocate memory for PI and assign a constant value

```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]){
    // Variable and constant declarations
    double radius, surfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere: ");
    radius = input.nextDouble();//assume
    // Step 2. Compute the surface area o
    surfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.pow(radius,2);

    // Step 3. Display the values of the radius and
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + surfaceArea + " sq cm\n";
 }
}
```

Memory Diagram

Variable name	Value/Content
radius	20
surfaceArea	
PI	3.14159

Assign the value to the variable radius after receiving it through the keyboard


```
/*******************************************************
*  This program finds the surface area                 *
*  of a sphere with a given radius                     *
*******************************************************/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]){
    // Variable and constant declarations
    double radius, surfaceArea;
    final double PI = 3.14159;

    // Step 1. Input the value of the radius
    Scanner input = new Scanner(System.in);
    System.out.print("Enter the radius of the sphere
    radius = input.nextDouble();//assume that th

    // Step 2. Compute the surface area of the s
    surfaceArea = 4 * PI * radius * radius;
    //Also: //SurfaceArea = 4 * Math.PI * Math.pow(radius,2);

    // Step 3. Display the values of the radius and
    // the surface area of the sphere.
    System.out.println(" The surface area of a sphere of radius "
    + radius + " cm, is " + surfaceArea + " sq cm\n";
 }
}
```

Memory Diagram

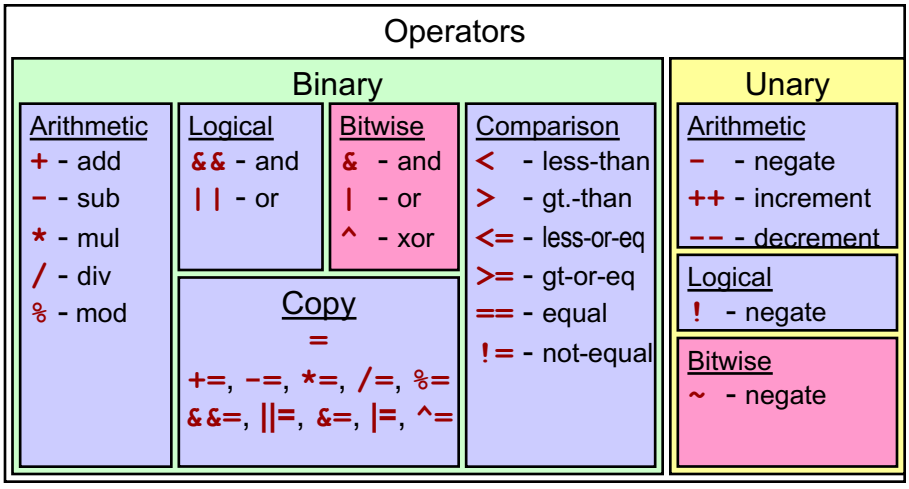
Variable name	Value/Content
radius	20
surfaceArea	5026.548245
PI	3.14159

Compute the surface area and assign it to variable surfaceArea

Java Operators

- Operators take one or more input values (operands) and produce one output value
  - E.g. `+` , `-` , `*` , `/` , `<` , `>` , `<=` , `>=` , `&&` , `||`
- Operators come in three flavours
  - Unary operators – take one operand (value)
  - Binary operators – take two operands (values)
  - Ternary operators – take three operands (values)

JavaOperator Map



Arithmetic Operators

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

Arithmetic Operators

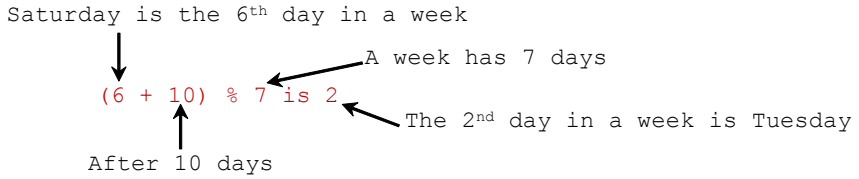
- `a+b`, `a-b`, `a*b`, `a/b`
- Integer division: `11/4` gives 2
- Floating point division:
  - `11.0/4` gives 2.75
- Modulo operator: `%`
  - `11%4.5` gives 2 (output the remainder after the division of 11/4.5)
  - `10%5` gives 0 (output the remainder after the division of 10/5)
  - `5.5%2.5` gives 0.5 (output the remainder after the division of 5.5/2.5)
  - `5%10` gives 5 (output the remainder after the division of 5/10)
  - **Practice problem**: How can we separate two digits from a two-digit decimal integer number, inputted from the keyboard, and print those digit on the screen (application of % and int-division)

Example: Average of three integer numbers

```
/* In-class work: Draw the flowchart for finding
  the average of three integers*/

import java.util.Scanner;
public class MyClass {
  public static void main(String args[]) {
     int number1 = 0, number2 = 0, number3 = 0;
     Scanner input = new Scanner(System.in);
     System.out.print("Input three different numbers: ");
     number1 = input.nextInt();
     number2 = input.nextInt();
     number3 = input.nextInt();
     double average = (number1+number2+number3)/3.0;
     System.out.println("The average is: " + average);
    }
}
```


The remainder/modulus operator

- Remainder is very useful in programming.
- For example, an even number % 2 is always 0 and an odd number % 2 is always 1. So you can use this property to determine whether a number is even or odd.
- Another Example: Suppose today is Saturday (6th day of the week) and you and your friends are going to meet in 10 days. What day is in 10 days? You can find that day is Tuesday using the following expression:



Example: application of % operator and integer division

```
/*Homework: Draw the flowchart for the following problem: Given a
   number of days (less than 365), compute the number of months and
   weeks; assume that each month is 30 days long, and the user will enter
   any integer number (no character or string)  less than 365*/
import java.util.Scanner;
public class MyClass {
    public static void main(String args[]) {
      int days = 0, months = 0, weeks = 0;
       Scanner input = new Scanner(System.in);
       System.out.print("Input number of days: ");
       days = input.nextInt();
       months = days/30;/*integer division will discard
       the fractional part*/
       days = days%30;//remaining days
       weeks = days/7;
       days = days%7; //remaining days
       System.out.println(months + " months, " + weeks
       + " weeks, and " + days + " days.");
    }
}
```


Practice Problem: Finding Time: Write a program that obtains minutes and remaining seconds from entered seconds.

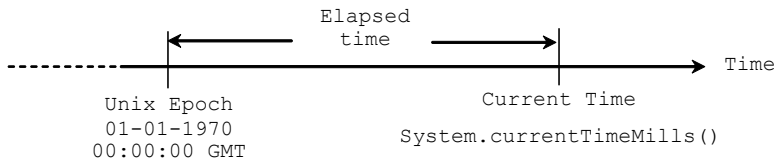
```
import java.util.Scanner;
public class DisplayTime {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    // Prompt the user for input
    System.out.print("Enter an integer for seconds: ");
    int seconds = input.nextInt();

    int minutes = seconds / 60; // Find minutes from seconds
    int remainingSeconds = seconds % 60; // Seconds remaining
    System.out.println(seconds + " seconds is " + minutes +
      " minutes and " + remainingSeconds + " seconds");
  }
}
```

Write a program that converts a Fahrenheit degree to Celsius using the formula: *Celsius* = (5/9)(*Fahrenheit* – 32).

```
//Watch out: you need to write: c = (5.0/9)*(f-32)

import java.util.Scanner;
public class FahrenheitToCelsius {
  public static void main(String[] args) {
    Scanner input = new Scanner(System.in);
    System.out.print("Enter a degree in Fahrenheit: ");
    double fahrenheit = input.nextDouble();
    // Convert Fahrenheit to Celsius
    double celsius = (5.0 / 9) * (fahrenheit - 32);
    System.out.println("Fahrenheit " + fahrenheit + " is " +
    celsius + " in Celsius");
  }
}
```


Practice Problem: Displaying Current Time

- Write a program that displays current time in GMT in the format hour:minute:second such as 1:45:19.
- Note: The currentTimeMillis method in the System class returns the current time in milliseconds since the midnight, January 1, 1970 GMT. (1970 was the year when the Unix operating system was formally introduced.)
- You can use this method to obtain the current time, and then compute the current second, minute, and hour as follows.



```
public class ShowCurrentTime {
  public static void main(String[] args) {
    // Obtain the total milliseconds since midnight, Jan 1, 1970
    long totalMilliseconds = System.currentTimeMillis();
    // Obtain the total seconds since midnight, Jan 1, 1970
    long totalSeconds = totalMilliseconds / 1000;
    // Compute the current second in the minute in the hour
    long currentSecond = totalSeconds % 60;
    // Obtain the total minutes
    long totalMinutes = totalSeconds / 60;
    // Compute the current minute in the hour
    long currentMinute = totalMinutes % 60;
    // Obtain the total hours
    long totalHours = totalMinutes / 60;
    // Compute the current hour
    long currentHour = totalHours % 24;
    // Display results
    System.out.println("Current time is " + currentHour + ":"
      + currentMinute + ":" + currentSecond + " GMT");
  }
}
```


Comparison Operators

Operator	Meaning	Example: assume a=1, b=2
>	Greater than	(a > b) returns false
<	Less than	(a < b) returns true
>=	Greater than or equal to	(a >= b) returns false
<=	Less than or equal to	(a <= b) returns true
==	Equal (equivalent)	(a == b) returns false
!=	Not equal	(a != b) returns true

Note: For very precise calculation avoid using == or != operators with floating-point numbers (double or float data types). double/float type numbers are stored in the memory using IEEE 754 floating point numbering system (http://en.wikipedia.org/wiki/IEEE_floating_point) and as a result .57 is stored as: .569999999999

Logical Operators

Operator	Meaning	Example of Use	Truth Value
&&	AND	(exp1) && (exp2)	Returns true if both exp1 and exp2 are logically true Returns false if any of exp1 or exp2 is logically false
||	OR	(exp1) || (exp2)	Returns true if any of exp1 or exp2 is logically true Returns false if both of exp1 and exp2 are logically false
!	NOT	! (exp1)	Returns true if exp1 is logically false Returns false if exp1 is logically true

- In this case, the expressions (exp1, exp2) have to be logically evaluated
- These are called "short-circuit" operators

Truth Tables for Logical Operators

AND (&&) Truth Table:

(exp1)	(exp2)	(exp1) && (exp2)
false	false	false
false	true	false
true	false	false
true	true	true

OR (||) Truth Table:

(exp1)	(exp2)	(exp1) || (exp2)
false	false	false
false	true	true
true	false	true
true	true	true

NOT (!) Truth Table:

(exp1)	!(exp1)
false	true
true	false

Logical Operators in expressions

```
(-6<0)&&(12>=10)
```

true && true results in *true*

```
(3.0 >= 2.0) || (3.0 >= 4.0)
```

true || false results in *true*

```
(3.0 >= 2.0) && (3.0 >= 4.0)
```

true && false results in *false*

Unary Operators

Given:
```
int a(9); boolean k = true;
```

- **Negate:** `-a` **gives -9**
- **Logical invert:** `!k` **gives** `false`
- **Increment:** `++`
  - `++a` **gives 10**
- **Decrement:** `--`
  - `--a` **gives 8**

More Unary Expressions

Given: `int a(9), b;`

- **Pre-Increment :**
  - `b = ++a;` b is 10 and a is 10
- **Post-Increment:**
  - `b = a++;` b is 9 and a is 10
- **Pre-Decrement :**
  - `b = --a;` b is 8 and a is 8
- **Post-Decrement:**
  - `b = a--;` b is 9 and a is 8

Expressions With Side Effects

- `r = x + y--;` is equivalent to two step operations

```
r = x + y; /*Step 1*/
y = y-1; /*Step 2*/
```

- `r = ++x - y;` is equivalent to two step operations

```
x = x+1; /*Step 1*/
r = x - y; /*Step 2*/
```

- Keep it simple
  - Not `r = --x + y++;`
- Compilation error!

```
cout<<(j+1)++<<endl; /*the incremented value is not
assigned to a memory location*/
```

- *Note: completely avoid writing any code that involves reassignment of a variable more than once on the same statement; such as r = –x + r++ where r = –x + r (step 1) and r = r+1 (step 2). Syntactically this is fine but some compilers may not accept it.*

Copy Operator (Assignment operator)

- Simple copy: `a = b;`
  - Overwrites the left object (l-value) with the result of the expression on the right (r-value)
  - l-value must be writable (not a const)
- Copy with add
  - `a += b;` is the same as `a = a + b;`
- Other copy-with-subtract, multiplication, division behaves the same
  - `a -= b; a *= b; a /= b`; etc

Operator Precedence

- Use parenthesis if you are not sure!
- Unary (`++(pre-increment)`, `--(pre-decrement)`) (left to right)
- Unary (`!`, `-`) (left to right)
- Arithmetic (`*`, `/`, `%`, `+`, `-`) (left to right) (BODMAS, for review: http://www.mathsisfun.com/operation-order-bodmas.html)
- Comparison (`<` , `<=` , `>` , `>=`, `==`, `!=`) (left to right)
- Logical binary (`&&`, `||`) (left to right; **if no bracket is used `&&` gets evaluated before `||`**)
- Assignment (`=` , `*=` , `/=` , `%=` , etc) (right to left)
- Unary (`++(post-increment)`, `--(post-decrement)`),

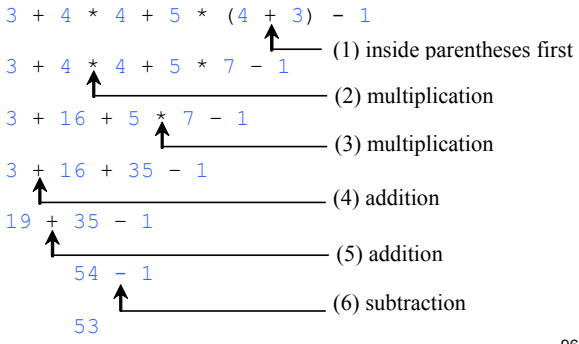
Note: U Are the best CLAss in the Universe

Numeric Operator Precedence

- **The numeric operators in C++ expression are applied as follows:**
  - **First operation: Operands contained with pre-increment or pre-decrement operators are evaluated first**
  - **Second operation: Operators contained within pairs of round parentheses are evaluated first**
    - **Parentheses (all are round) can be nested and , in thisS case the inner one is evaluated first**
  - **Third operations: (multiplications, division and modulus) If the expression contain several multiplication, division and modulus operations, they are applied from left to right (higher precedence than addition and subtraction)**
  - **Fourth operations: If an expression contains several addition and subtraction, they are applied from left to right**
  - **Fifth operations: Assignment operations**
  - **Last operations: post-increment or post-decrement operations**

Evaluating an expression in Java

- Note: In the following expression, each of the operators is a binary operator, and hence each operation is performed using two operands at a time.
- Watch for the BODMAS rule, and left to right rule.



Expressions

- A series of operators and their inputs
  - E.g. `x+3-y+5`
- Evaluated from left-to-right
- Be careful with division
- Pay attention to operator precedence

$$\frac{x + 3}{5} + \frac{4}{y + 3}$$

is not the same as

```
x + 3 / 5 + 4 / y + 3
```

Correct form:

```
(x+3)/5 + 4/(y+3)
```


Expression Examples

- `2*x*x-3*y/5*y+6`
- or `2*x*x-(3*(y/5)*y)+6`

$$\Rightarrow 2x^2 - \frac{3y}{5}y + 6$$

- `(2*x*x-3*y)/(5*y+6)`

$$\Rightarrow \frac{2x^2 - 3y}{5y + 6}$$

Expression Example

The following expression:

$$z = \frac{3+4x}{5} - \frac{10(y-5)(a+b+c)}{x} + 9\left(\frac{4}{x} + \frac{9+x}{y}\right)$$

is written in java syntax as:

```
Z=(3+4*x)/5 - 10*(y-5)*(a+b+c)/x + 9*(4/x + (9+x)/y)
```


Characters and String

- **char** type variable holds a single character.
  example:

```
char alphabet = 'k';
```

- **String** type object (variable) holds a sequence of characters
  example:

```
String title = "Java programming";
```

Note: string is NOT a data type; it's a class-type (discussed later) which can be used as a data type.

String Operators

- **= operator assigns a value to a string**

```
string taste;
taste = "Yummy ";
```

- **+ operator joins two strings together**

```
string s1 = "hot", s2 = "dog";
string food = s1 + s2;
// food = hotdog
```

- **+= operator for concatenation**

```
taste += food;
// taste = Yummy hotdog
```


Backslash Codes with print()/println() method

code	Meaning
\n	**new line**
\t	**horizontal tab**
\"	**double quote**
\'	**single quote**
\\	**backslash**
\b	**backspace**
\v	**vertical tab** (when output to printer)
\f	**form feed** (when output to printer)
\r	**carriage return**

- These codes are also known as "escape sequence".
- These escape sequences need to be used as string constants (enclosed in a pair of double quotation marks)

Practice problem: Output?

```
public static void main(String[] arg)
{
  System.out.print("\\\n\\\\\n\\\\\\\n");
}
```


Math methods in java.lang.Math Class

https://docs.oracle.com/javase/10/docs/api/java/lang/Math.html
All are static methods; use Math.methodname().

`abs(x)`	computes absolute value of x
`sqrt(x)`	computes square root of x, where x >=0
`pow(x,y)`	computes x^y
`ceil(x)`	nearest integer larger than x
`floor(x)`	nearest integer smaller than x
`exp(x)`	computes e^x
`log(x)`	computes ln x, where x >0
`log10(x)`	computes $\log_{10}x$, where x>0
`max(x,y)`	Returns the greater of the two arguments
`min(x,y)`	Returns the smaller of the two arguments
`floorMod(x,y)`	Returns the floor modulus of the arguments

Math methods in java.lang.Math Class

`sin(x)`	sine of x, where x is in radians
`cos(x)`	cosine of x, where x is in radians
`tan(x)`	tangent of x, where x is in radians
`asin(x)`	$sine^{-1}(x)$, returns angle in radians $[-\pi/2, \pi/2]$
`acos(x)`	$cosine^{-1}(x)$, returns angle in radians $[0,\pi]$
`atan(x)`	$tan^{-1}(x)$, returns angle in radians $[-\pi/2, \pi/2]$
`atan2(y,x)`	$tan^{-1}(y/x)$, returns angle in radians $[-\pi, \pi]$; the angle value after conversion from rectangular to polar coordinate.
`sinh(x)`	Hyperbolic sine of x
`cosh(x)`	Hyperbolic cosine of x
`tanh(x)`	Hyperbolic tan of x
`toDegrees(x)`	Converts x radians to degrees
`toRadians(x)`	Converts x degrees to radians

Some Examples on Math methods

- //Dispalying sin(45 deg) = 0.7071067811865475
  System.out.println("sin(45 deg) = "+Math.sin(Math.toRadians(45)));
- //Displaying 5.2^-3 = 0.007111970869367318
  System.out.println("5.2^-3 = "+Math.pow(5.2,-3));
- //Dsipalying a larger number between 5.92 and 5.919
  System.out.println(Math.max(5.92,5.919));
- //Dsipalying log (base 10) of a number
  System.out.println(Math.log10(100));
- //Dsipalying natual log of a number
  System.out.println(Math.log(100));

Arithmetic conversions

Assigning a value of one type to another requires one of the following operations:

- Implicit Conversion
- Explicit Conversion

Implicit Conversion (IC)

- In an arithmetic expression, the final result of the expression (with different numerical data types) will be stored using a data type of the operand (present in the expression) having the highest data range in the memory.
- Implicit arithmetic conversion is also called widening, automatic conversion, implicit conversion, coercion or promotion.

  `byte→ short→int→long→float→double`

  Widening → …→ …→ …→ …→ …→
- Ex: *int a*, *float b*; (*a* + *b*) will be a *float* type where *int a* gets promoted/widened to *float a*.

Explicit Conversions: Casting

- Explicit arithmetic conversions are called *casting*. Also known as narrowing, because it is required for narrowing operation only.
  - For example, *int a* can be explicitly converted to a float variable in any statement that follows the above declaration, by the cast operator as: (*float*) *a.*
- Cast operator: (data_type) variable
- Cast operator is an unary operator
- Cast operator works only for the part of the statement that uses it. Example:

```
int x = 5;
System.out.println(x/2);//output?
System.out.println((float)x/2 + x/10);//output?
System.out.println(x/2);//output?
```

Narrowing: double→float→long→int→short→byte

Program Example with Widening

```
public class MyClass {
 public static void main(String args[]) {
  int i = 100;
  long l = i;//no explicit type casting required
  float f = l;//no explicit type casting required
  System.out.println("Int value "+i);
  System.out.println("Long value "+l);
  System.out.println("Float value "+f);
 }
}
```

Program Example with Narrowing

```
public class MyClass {
 public static void main(String args[]) {
  double d = 100.04;
  long l = (long)d;//explicit casting required
  int i = (int)l;//explicit casting required
  System.out.println("Double value "+d);
  System.out.println("Long value "+l);
  System.out.println("Int value "+i);
 }
}
```


Practice Problem: Computing Loan Payments

- This program lets the user enter the interest rate, number of years, and loan amount, and computes monthly payment and total payment.

(Solution on the next slide: Check the Math method used there. Also, check the application of casting)

$$monthlyPayment = \frac{loanAmount \times monthlyInterestRate}{1 - \frac{1}{(1 + monthlyInterestRate)^{numberOfYears \times 12}}}$$

```
import java.util.Scanner;
public class ComputeLoan {
  public static void main(String[] args) {
    // Create a Scanner
    Scanner input = new Scanner(System.in);
    // Enter yearly interest rate
    System.out.print("Enter yearly interest rate, for example 8.25: ");
    double annualInterestRate = input.nextDouble();
    // Obtain monthly interest rate
    double monthlyInterestRate = annualInterestRate / 1200;
    // Enter number of years
    System.out.print("Enter number of years as an integer, for example 5: ");
    int numberOfYears = input.nextInt();
    // Enter loan amount
    System.out.print("Enter loan amount, for example 120000.95: ");
    double loanAmount = input.nextDouble();
    // Calculate payment
    double monthlyPayment = loanAmount * monthlyInterestRate / (1
      - 1 / Math.pow(1 + monthlyInterestRate, numberOfYears * 12));
    double totalPayment = monthlyPayment * numberOfYears * 12;
    // Display results
    System.out.println("The monthly payment is $" + (int)(monthlyPayment * 100) / 100.0);
    System.out.println("The total payment is $" + (int)(totalPayment * 100) / 100.0);
  }
}
```


Literal and Default data-type

- Any literal of whole number is an integer (not byte or long) by default
- Any literal of fractional number is double type (not float) by default
- Example:

```
System.out.println( 25/3.0); //prints 8.333333333333334
//25 is an integer literal, and 3.0 is double type literal.
```

- To denote a literal of whole number as long type, append it with the letter L or l.
- To denote a literal of fractional number as float type, append it with the letter F or f.
- Example:

```
System.out.println( 25L/3.0F); //prints 8.333333
//25 is a long-type literal, and 3.0 is float type literal.
```

Review

12) `a+b/c-d` is the same as

a) a+b/(c-d)
b) a+(b/c)-d
c) (a+b)/c-d
d) (a+b)/(c-d)

Review

13) Which of the following expressions will result in a zero value?

a) 8%3 - 1
b) 3 - 8%5
c) 9%3 - 1
d) 2 - 5%2

Review

14) Which Java expression computes the following equation correctly?

$$\frac{a(x-y)}{b} + c$$

a) `(a/b)*x-y+c`

b) `a/(b*(x-y))+c`

c) `a/(b*x-y)+c`

d) `(a/b)*(x-y)+c`

Review

15) What is the value of newNum after these two statements?

```
int i = 10;
int newNum = 10 * i++;
```

a) 10

b) 100

c) 110

d) 1000

Review

16) The following two Java statements perform the same operation.

```
regWages = regPay + overTime;
regPay + overTime = regWages;
```

a) True
b) False

Review

17) The following two expressions will result in the same value:

```
a + b * c
b * c + a
```

a) True
b) False

Review

18) The following statement doubles the value stored in answer

```
answer *= 2;
```

a) True
b) False

Review

19) Which of the following is the correct expression that evaluates to true if the number x is in between 1 and 100 or the number is negative?

```
a) (1 < x) || (x < 100) && (x < 0)
b) ((x < 100) && (x > 1)) || (x < 0)
c) ((x < 100) && (x > 1)) && (x < 0)
d) (1 > x) && (x > 100) || (x < 0)
```

Review

20) What will be the value of 'c' after the third statement? Let's draw the memory diagram (in class discussion)

```
int a = 3, b = 2;
float c;
c = (a+b)/2;
```

a) 2.5
b) 2.0
c) 3.0
d) 0

Review

21) What will be the output of the code? Let's draw the memory diagram (in class discussion)

```
int a = 2, b = 4;
a = ++b;
System.out.println(a+", "+b);
```

a) 2, 5
b) 4, 4
c) 5, 5
d) 4, 5

Review

22) value of variable 'a' after the second statement?

```
boolen a = 2;
a = 5 == 5;
System.out.println(a);
```

a) 1
b) true
c) false
d) 0

Review

23) Output ?

```
int a;
System.out.println(a);
```

a) 0
b) Any random value
c) Compilation error will be generated
d) Run time error will be generated

Review

24) Output ?

```
int a = 2, b = a+2; double
b(4);
System.out.println(a/b);
```

a) 0
b) 0.5
c) The code will result in a Compilation error
d) The code will result in a Run time error

Common Errors and Pitfalls

- Common Error 1: Undeclared/Noninitialized Variables
- Common Error 2: Integer Overflow
- Common Error 3: Round-off Errors
- Common Error 4: Unintended Integer Division
- Common Pitfall 1: Redundant Input Objects

Common Error 1: Undeclared/Noninitialized Variables

```
// Undeclared variable
double hourlyRate = 13.50;
double yourPayment = hourlyrate * 35;
// NonInitialized variable
int itemsPerBox, box = 11;
int totalItems = itemsPerBox *box;

//Result: Compilation Error
```


Common Error 2: Integer Overflow

```
int k = 2147483647+1;
System.out.println(k);
//output: -2147483648
//Result: logical Error; check the range for int

Note: since int is a default type, logical
error will be generated, but for other type
(short, byte, long) going out of rang will
result in a compilation error unless we cast
it.
```

Common Error 3: Undeclared/Noninitialized Variables

```
System.out.println(1.0-0.1-0.1-0.1-0.1-0.1);
// Prints: 0.5000000000000001
//Result: Logical Error
System.out.println(1.0-0.9);
// Prints: 0.09999999999999998
//Result: Logical Error
```


Common Error 4: Unintended Integer Division

```
int number1=3;
int number2=2;
System.out.println("Average = "+(number1 + number2)/2);
// Prints: 2
//Result: Logical Error
======== Below, the error is fixed ==========
int number1=3;
int number2=2;
System.out.println("Average = "+(number1 + number2)/2.0);
// Also, one of the following three will work
System.out.println("Average = "+(number1 + number2)/2D);
System.out.println("Average = "+(number1 + number2)/2F);
System.out.println("Average = "+(number1 + number2)/(double)2);
```

Common Pitfall 1: Redundant Input Objects

- Only one scanner object can be used to input any type of data from the standard input:

Example:

```
Scanner input = new Scanner(System.in);
//Declaring second scanner object is redundant:
Scanner inputD = new Scanner(System.in);
System.out.print("Input a number: ");
int intNumber = input.nextInt();
double doubleNumber = inputD.nextDouble();
//Instead of the above statement, the following can be used:
double doubleNumber = input.nextDouble();
```


Programming Style and Documentation

This is your reading assignment. This part is very important for code-writing and labs

- Appropriate Comments
- Naming Conventions
- Proper Indentation and Spacing Lines
- Block Styles

Appropriate Comments

- Include a summary at the beginning of the program to explain what the program does, its key features, its supporting data structures, and any unique techniques it uses.
- Include your name, class section, instructor, date, and a brief description at the beginning of the program.
- Include comments for any statements which may result in confusion for you when you are reviewing your own code later.

Naming Conventions

- Choose meaningful and descriptive names. To store the radius value, choose a name `radius` instead of `x` or `y`.
  - Note: If you are calculating an expression such as: `y = x^2 + 3z`, then it will be a good idea to choose the same variable names `x`, `y` and `z` in the program.
- Variables and method (function) names:
  - Use lowercase. If the name consists of several words, concatenate all in one, use lowercase for the first word, and capitalize the first letter of each subsequent word in the name.
    - For example, the variables `interestRate` and `year`, and the function `computePayment`.

Naming Conventions, *cont.*

- Class names (*Class will be discussed later*):
  - Capitalize the first letter of each word in the name. For example, the class name `InterestRateCalculation`.
- Constants:
  - Capitalize all letters in constants, and use underscores to connect words. For example, the constant `MAX_VALUE` or `AGE`.

Proper Indentation and Spacing

- Indentation
  - Indent two spaces.
- Spacing
  - Use blank line to separate segments of the code.

Block Styles

Use either next-line (next to the heading line) or end-of line style for braces (next line below to the heading line). The heading line is known as 'header'.

```
public static void main()
{//Next line style
      System.out.println("This is next-line
      block style");
}
```

```
public static void main(){//end-of-line style
      System.out.println("This is end-of-line
      block style");
}
```


Displaying output using Formatted Output (Labs only)

Note: We can display any formatted output using printf/sprintf statements as in C

```
public class DisplayTime {
  public static void main(String[] args) {
    int seconds = 500;
    int minutes = seconds / 60;
    int remainingSeconds = seconds % 60;
    System.out.printf("%d seconds is %d
minutes and %d remaining Seconds\n",seconds,
minutes, remainingSeconds);
  }
}
```

The *printf* method (Labs only)

- The *printf* method is used to send text and numbers to the standard output (screen).

- The general form of the *printf* method is:

```
public PrintStream printf(control_string,
  other_arguments);
```

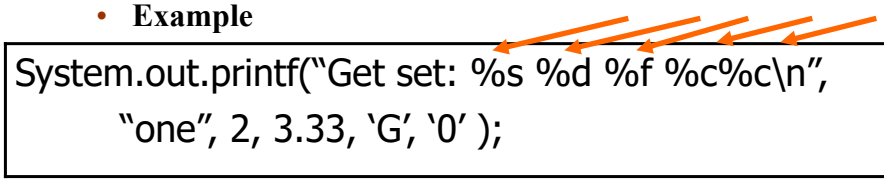

Frequently-Used Format Specifiers

Specifier	Output	Example
%b	a boolean value	true or false
%c	a character	'a'
%d	a decimal integer	200
%f	a floating-point number	45.460000
%e	a number in standard scientific notation	4.556000e+01
%s	a string	"Java is cool"
%o	An octal number	12
%x	A hexadecimal number	

Note: Format specifiers are required for printf/safprintf statements

Format Specifiers: Examples

- **Example**



```
System.out.printf("Get set: %s %d %f %c%c\n",
        "one", 2, 3.33, 'G', '0' );
```

Get set: one 2 3.330000 GO

Format Specifiers: Examples

- **The number of digits after the decimal point (precision) can be specified for floating point values**

```
System.out.printf("Get set: %s %d %.2f %c%c",
        "one", 2, 3.33426, 'G', '0' );
```

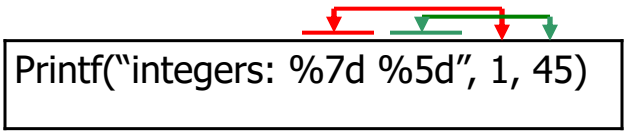
Get set: one 2 3.33 GO

Field and Field width (Labs only)

- **Control string specifies where the arguments will be placed on the screen.**
- **Arguments are placed on a block of spaces on the screen called a *field*.**
- **The number of spaces, called *field width***
- **The arguments can be right-justified (by default) or left justified.**

Control string: Examples

- Field width and right justification (default)



- **Integers:** | 1 | 45 |

Field width = 7
(six spaces and 1)

Field width = 5
(three spaces and 45)

Control string: Examples

- Field width and right justification

```
printf("floats: %7.1f%6.2f", 1.0, 45.26);
```

floats: 1.0 45.26

Field width = 7
(four spaces and 1.0)

Field width = 6
(one space and 45.26)

Control string: Examples

- Field width and left justification
- Left justification is achieved by adding a – sign after the % sign.

```
printf("floats: %-7.1f%-6.2f", 1.0, 45.26);
```

- floats: 1.0 45.26

Field width = 7
(1.0 and four spaces)

Field width = 6
(45.26 and one space)

NOTE

- Calculations involving floating-point numbers are approximated because these numbers are not stored with complete accuracy. For example,
- System.out.println(1.0 - 0.1 - 0.1 - 0.1 - 0.1 - 0.1);

  displays 0.5000000000000001, not 0.5, and
- System.out.println(1.0 - 0.9);

  displays 0.09999999999999998, not 0.1. Integers are stored precisely. Therefore, calculations with integers yield a precise integer result.

Practice problem: Given an amount (less than $1.00), compute the number of quarters, dimes, nickels and pennies needed to make up that amount. Your code should pick up the highest valued coin first and then gradually go down to the lowest valued one.

Answers to the review questions

1. A
2. C
3. A
4. A
5. B
6. A
7. B
8. B
9. D
10. C
11. D
12. B
13. B
14. D
15. B
16. B
17. A
18. A
19. B
20. B
21. C
22. B
23. C
24. C