



SWENG3101

Object Oriented Programming

Department of Software Engineering

AASTU

November, 2021

Chapter 2

Basics of Java and Control Statements

Outline

- Basics of Java Programming
- Conditional and Loops
- Arrays
- Core Classes of Java

Definition and History of java

- Java is an object-oriented programming language.
- Java was developed by **James Gosling** and his team at **Sun Microsystems** in California. Sun formally announced Java at an industry conference in May 1995.
- In 1991 Sun funded an internal corporate research project code-named **Green**. The language was based on C and C++ and was originally intended for writing **programs that control consumer appliances** such as toasters, microwave ovens, and others.
- The language was first called **Oak**, named after the oak tree outside of Gosling's office, but the name was already taken, so the team renamed it Java.

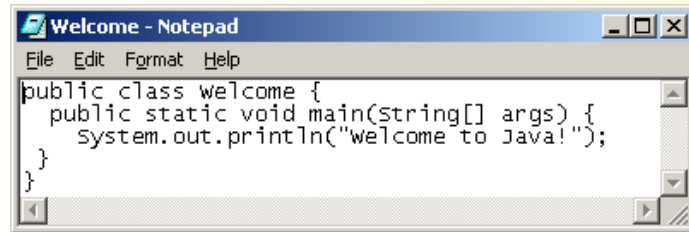
Java Application and Java Applet

- Java can be used to program two types of programs: Applets and applications.
- **Applets:** are programs called that run within a Web browser. That is, you need a Web browser to execute Java applets.
- Applets allow more dynamic and flexible dissemination of information on the Internet
- **Java Application:** is a complete stand-alone program that does not require a Web browser.
- A Java application is analogous to a program we write in other programming languages.

Typical Java Environment

- Java programs normally undergo five phases
 - Edit
 - Programmer writes program (and stores program on disk)
 - Compile
 - Compiler creates **bytecodes** from program
 - Load
 - Class loader stores bytecodes in memory
 - Verify
 - Verifier ensures bytecodes do not violate security requirements
 - Execute
 - Interpreter translates bytecodes into machine language

Cont'd...



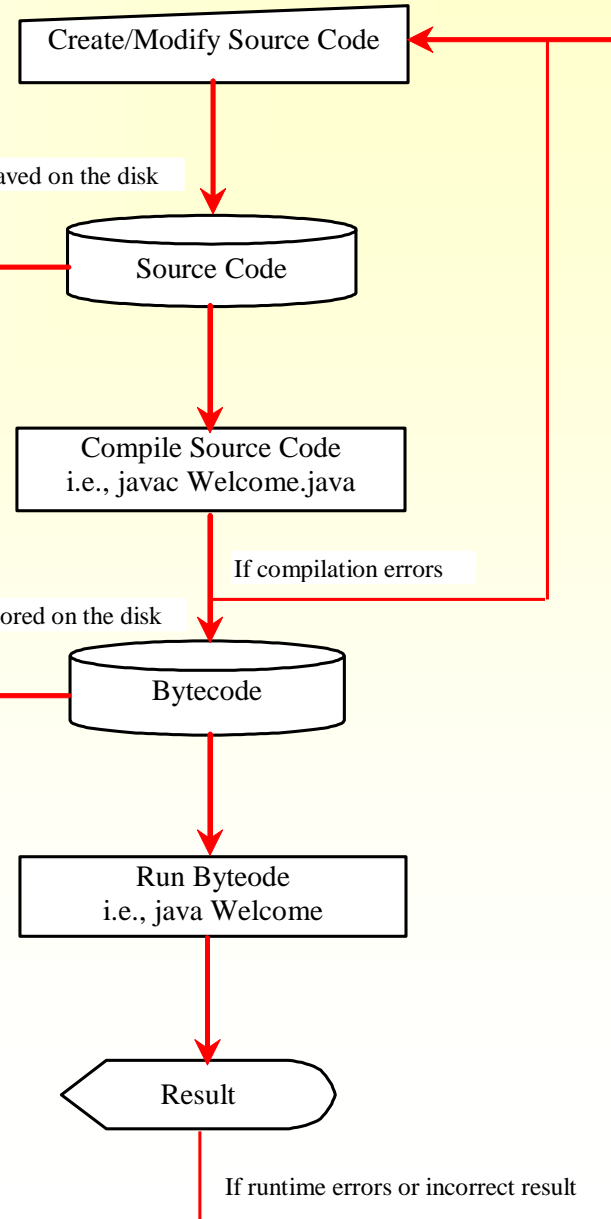
```
File Edit Format Help
public class welcome {
    public static void main(String[] args) {
        System.out.println("welcome to Java!");
    }
}
```

Source code (developed by the programmer)

```
public class Welcome {
    public static void main(String[] args) {
        System.out.println("Welcome to Java!");
    }
}
```

Byte code (generated by the compiler for JVM
to read and interpret, not for you to understand)

```
...
Method Welcome()
  0 aload_0
  ...
Method void main(java.lang.String[])
  0 getstatic #2 ...
  3 ldc #3 <String "Welcome to
  Java!">
  5 invokevirtual #4 ...
  9 return
```

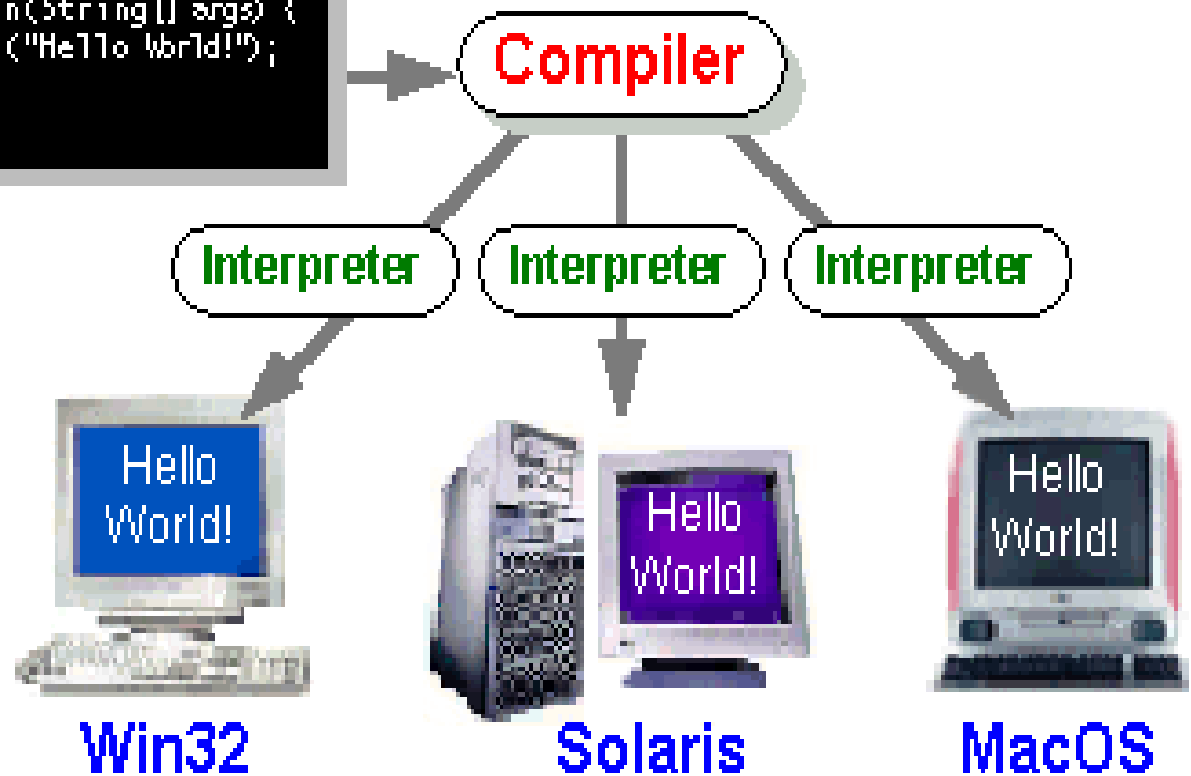


Cont'd

Java Program

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

HelloWorldApp.java



Typical Java Environment(cont'd)

■ ***Phase 1: Creating a Program***

- consists of editing a file with an **editor program (editor)**.
- You type a Java program (**source code**) using the editor, make any necessary corrections and save the program.
- A file name ending with the **.java extension** indicates that the file contains Java source code

■ ***Phase 2: Compiling a Java Program into Bytecodes***

- Use the command **javac** (the **Java compiler**) to **compile** a program.
- ***E.g.: To compile a program called `Welcome.java`, you would type `javac Welcome.java` in the command window of your system***

Typical Java Environment(cont'd)

- If the program compiles, the compiler produces a **.class file** called **Welcome.class** that contains the compiled version of the program.
- The Java compiler translates Java source code into **bytecodes** that represent the tasks to execute in the execution phase (Phase 5).
- Byte codes are executed by the **Java Virtual Machine (JVM)**.
- **Virtual machine (VM)** is a software application that simulates a computer, but hides the under-lying operating system and hardware from the programs that interact with the VM.
- Java's bytecodes are **portable**. Unlike machine language, which is dependent on specific computer hardware, bytecodes are platform-independent instructions. That is why Java is guaranteed to be **Write Once, Run Anywhere**.
- The JVM is invoked by the **java** command.
- For example, to **execute** a Java application called Welcome, you would type the command **java Welcome** in a command window **to invoke the JVM**, which would then initiate the steps necessary to execute the application. **This begins Phase 3.**

Typical Java Environment(cont'd)

■ Phase 3: **loading**: *placing the program in memory before executing it.*

- The **class loader** takes the **.class** files containing the program's bytecodes and transfers them to primary memory.
- The **.class** files can be loaded from a disk on your system or over a network (e.g., your local college or company network, or the Internet).

■ **Phase 4: Bytecode Verification**

- **bytecode verifier** examines their bytecodes to ensure that they are valid and do not violate Java's security restrictions.
- Java enforces strong security, to **make sure that Java programs arriving over the network do not damage your files or your system** (as computer viruses and worms might).

Typical Java Environment(cont'd)

■ **Phase 5: Execution**

- The JVM executes the program's bytecodes, thus performing the actions specified by the program.
- In early Java versions, the JVM would interpret and execute one bytecode at a time; **Java programs execute slowly.**
- Today's JVMs typically execute bytecodes using a combination of interpretation and so-called **just-in-time (JIT) compilation.**
 - The JVM analyzes the bytecodes as they are interpreted, searching for **hot spots**— parts of the bytecodes that execute frequently.

■ *Integrated development environment (IDE)* combines these steps into a single task.

- A graphical user interface is available for editing, compiling, linking, and running the application.
- Example: NetBeans, Eclipse, Jbuilder, etc.

Activity

- Describe:
 - Compile time error
 - Run-time error
 - Logical error
- What is Java bytecode?

Java Features

- Object Oriented
- Simple
 - Java is partially modeled on C++, but greatly simplified and improved
- Architectural-neutral: Java compiler generates an architecture-neutral object file format which makes the compiled code to be executable on many processors
 - Platform independent
 - Portable
 - Java promise: “Write once, run everywhere”.
- Multithreaded
 - possible to write programs that can do many tasks simultaneously.

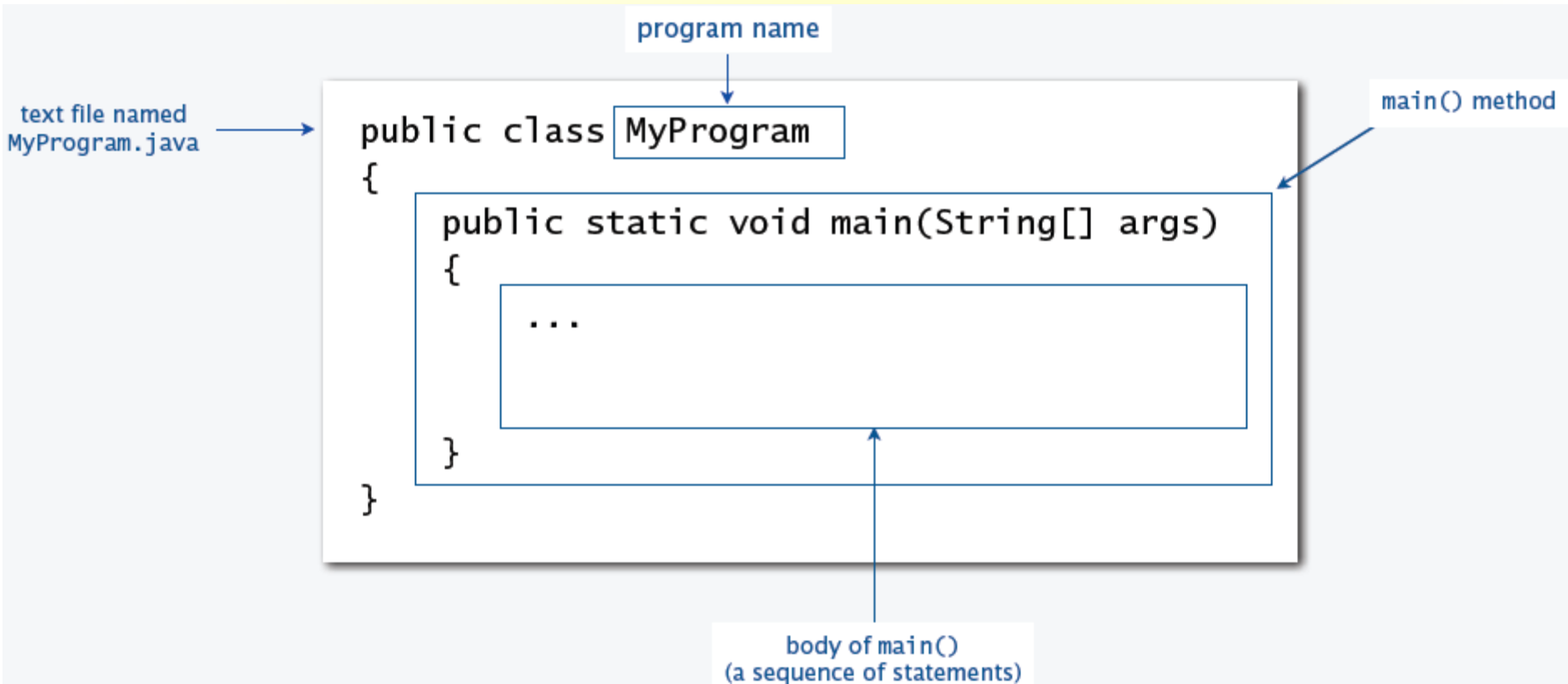
Java Features(cont'd)

- **Robust**: measures how well the program runs under various conditions.
 - Java compilers can detect many problems that would first show up at execution time in other languages.
 - Java has a **runtime exception-handling** feature to provide programming support for robustness.
- **Distributed**
 - Java applications can open and access objects across the Net via URLs with the same ease as when accessing a local file system.
- Both **compiled** and **interpreted** language.
- **High Performance**: With the use of **Just-In-Time compilers**, Java enables high performance.
- **Secure**

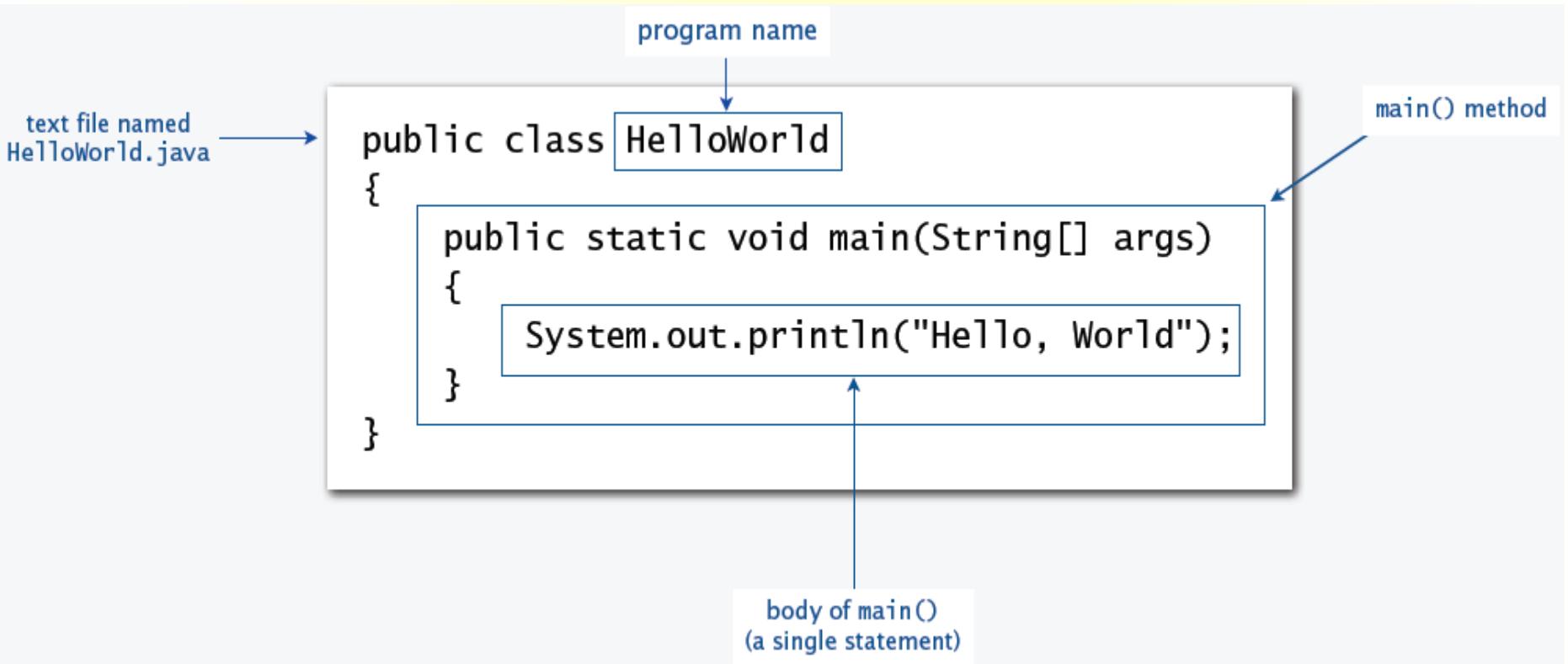
Java Program Structure

- In the Java programming language:
 - A program is made up of one or more *classes*
 - A class contains one or more *methods*
 - A method contains program *statements*
- A Java application always contains a method called `main`
 - Java program processing starts from the `main()`

Java Program Structure(cont'd)



Java Program Structure(cont'd)



First Java Program

- A simple code that would print the words Hello World.

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

Class Names and Identifiers

- Java is a case sensitive **E.g. Hello** is different from **hello**
- **Class Names** - For all class names the first letter shall be in Upper Case. If several words are used to form a name of the class, each inner word's first letter should be in Upper Case.
 - Example class MyFirstJavaClass
- **Method Names** - All method names shall start with a lower case letter. If several words are used to form the name of the method, then each inner word's first letter should be in Upper Case.
 - Example: `public void myMethodName()`
- **Program File Name** - Name of the program file **should** exactly match the class name.
 - Example: Assume 'MyFirstJavaProgram' is the class name. Then the file should be saved as 'MyFirstJavaProgram.java'.

Cont'd

- Names used for classes, variables and methods are called **identifiers**.
 - All identifiers should begin with a letter (A to Z or a to z), currency character (\$) or an underscore (_).
 - After the first character identifiers can have any combination of letters, digit, \$, and _.
 - A key word cannot be used as an identifier.
 - Identifiers are case sensitive.
 - Examples of legal identifiers: age, \$salary, _value, __1_value
 - Examples of illegal identifiers: 123abc, -salary

Java Keywords

- The following list shows the reserved words in Java. These reserved words may not be used as constant or variable or any other identifier names.


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

Comments in Java

■ E.g.

```
public static void main(String []args){  
    // This is an example of single line comment  
    /* This is also an example of  
        multiline comment. */  
        System.out.println("Hello World");  
    }  
}
```

Java language vocabulary

<i>built-in types</i>	<i>operations on numeric types</i>	<i>String operations</i>	<i>assignment</i>	<i>object oriented</i>	<i>Math methods</i>	
int	+	+	=	static	Math.sin()	
long	-	""		class	Math.cos()	
double	*	length()	<i>flow control</i>	public	Math.log()	
char	/	charAt()	if	private	Math.exp()	<i>System methods</i>
String	%	compareTo()	else	new	Math.pow()	System.print()
boolean	++	matches()	for	final	Math.sqrt()	System.println()
	--		while	toString()	Math.min()	System.printf()
		<i>boolean operations</i>		main()	Math.max()	
<i>punctuation</i>	<i>comparisons</i>		<i>arrays</i>		Math.abs()	<i>our Std methods</i>
{	<	true	a[]		Math.PI	StdIn.read*()
}	<=	false	length	<i>type conversion methods</i>		StdOut.print*()
(>	!		Integer.parseInt()		StdDraw.*()
)	>=	&&		Double.parseDouble()		StdAudio.*()
,	==					StdRandom.*()
;	!=					

Your programs will primarily consist of these plus identifiers (names) that you make up

Variable

- A *variable* is a name for a location in memory
- You must declare all variables before they can be used. The basic form of a variable declaration is:
 - data type variable [= value][, variable [= value] ...] ;

data type

variable name

`int total;`

- E.g. `int a, b, c; // Declares three ints, a, b, and c.`
 - `int a = 10, b = 10; // Example of initialization`
 - `byte B = 22; // initializes a byte type variable B.`
 - `double pi = 3.14159; // declares and assigns a value of PI.`
 - `char a = 'a'; // the char variable a is initialized with value 'a'`
 - `char tm = '\u2122'; // \u2122 is the trademark symbol`

Data Types

- There are two data types available in Java:
 - Primitive Data Types
 - Reference/Object Data Types

Data Type	Size	Range
byte	1 byte	Integers in the range of -128 to +128
short	2 bytes	Integers in the range of -32,768 to +32,767
int	4 bytes	Integers in the range of -2,147,483,648 to +2,147,483,647
long	8 bytes	Integers in the range of -9,223,372,036,854,775,808 to +9,223,372,036,854,775,807
float	4 bytes	Floating-point numbers in the range of $\pm 3.4 \times 10^{-38}$ to $\pm 3.4 \times 10^{38}$ with 7 digits of accuracy
double	8 bytes	Floating-point numbers in the range of $\pm 1.7 \times 10^{-308}$ to $\pm 1.7 \times 10^{308}$ with 15 digits of accuracy
char	2 bytes	A single character
Boolean	1 byte	true or false

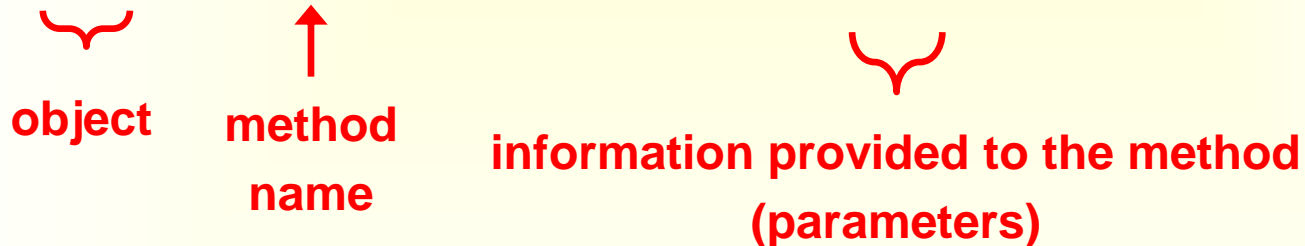
Constants

- A constant is an identifier that is similar to a variable except that it holds the same value during its entire existence
- Example:
 - `byte a = 68;`
 - `char a = 'A'`
- In Java, we use the `final` modifier to declare a constant
 - `final int MIN_HEIGHT = 69;`

print and println Methods

- We can invoke the `println` method to print a character string
- The `System.out` object represents a destination (the monitor screen) to which we can send output

```
System.out.println ("Whatever you are, be a good one.");
```


object **method name** **information provided to the method (parameters)**

- The `print` method is similar to the `println` method, except that it does not advance to the next line
- Therefore anything printed after a `print` statement will appear on the same line

Escape Sequence

- What if we wanted to print a the quote character?
- The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println ("I said "Hello" to you.");
```

- An *escape sequence* is a series of characters that represents a special character
- An escape sequence begins with a backslash character (\)

```
System.out.println ("I said \"Hello\" to you.");
```

Escape Sequence

Notation	Character represented
\n	Newline (0x0a)
\r	Carriage return (0x0d)
\f	Formfeed (0x0c)
\b	Backspace (0x08)
\s	Space (0x20)
\t	tab
\"	Double quote
\'	Single quote
\\	backslash
\ddd	Octal character (ddd)
\uxxxx	Hexadecimal UNICODE character (xxxx)

EscapeExample.java

```
public class EscapeExample{  
    public static void main( String[] args){  
        System.out.println("Hello World");  
        System.out.println("two\nlines");  
        System.out.println("\"This is in quotes\"");  
    }  
}
```

Output:

Hello World

two

lines

“This is in quotes”

String

- A string of characters can be represented as a *string literal* by putting double quotes around the text:
- Examples:
 - `"This is a string literal."`
 - `"123 Main Street"`
- There are close to 50 methods defined in the String class. We will introduce some of them here:
- **substring**
 - E.g. String text;
text = "Espresso";
System.out.print(text.substring(2, 7)); //char at end position is not included
 - **Output: press**
- **length**
 - **Example**
 - text1 = ""; //empty string
 - text2 = "Hello";
 - text3 = "Java";
 - text1.length() **//output: 0**
 - text2.length() **//output: 5**
 - text3.length() **//output: 4**

String(cont'd)

■ equals

- Returns true if the calling object string and the *Other_String* are equal. Otherwise, returns false.
- Example
 - String greeting = "Hello";
 - greeting.equals("Hello") returns true
 - greeting.equals("hello") returns false

■ toLowerCase

- Convert to lowercase
 - String greeting = "Hi Mary!";
 - greeting.toLowerCase() returns "hi mary!".

■ toUpperCase

- Convert to uppercase
 - String greeting = "Hi Mary!";
 - greeting.toUpperCase() returns "HI MARY!".

String(cont'd)

■ Indexof

- Locate the index position of a substring within another string

■ Example

- `text = "I Love Java and Java loves me.";`
- `text.indexOf("J") = 7`
- `text.indexOf("love") = 21`
- `text.indexOf("ove") = 3`

■ String concatenation

- Use the plus symbol (+) for string concatenation

- `string text1 = "Jon";`
- `string text2 = "Java";`
- `text1 + text2----- "JonJava"`
- `text1 + " " + text2----- "Jon Java"`
- `"How are you, " + text1 + "?"----- "How are you, Jon?"`

StringExample.java

```
public class StringExample{  
    public static void main(String[] args) {  
        String text1="Hello", text2 = "World";  
        String text3 = "I Love Java and Java loves me.";  
        System.out.println(text1.substring(1,3));  
        System.out.println(text2.length());  
        System.out.println(text3.indexOf("I"));  
        System.out.println(text1+" "+text2);  
    }  
}
```

Output:?

Type Conversion/Casting

- A value in any of the built-in types we have seen so far can be converted (type-cast) to any of the other types.

- For example:

`(int) 3.14` `// converts 3.14 to an int 3`

`(double) 2` `// converts 2 to a double to give 2.0`

- **Integer division always results in an integer outcome.**

- E.g. $9/2$ gives 4 not 4.5

- E.g. $8.2/2=4.1$ implicit conversion/promotion

Java Basic Operators

- Java provides a rich set of operators to manipulate variables. We can divide all the Java operators into the following groups:
 - Arithmetic Operators
 - Relational Operators
 - Logical Operators
 - Assignment Operators

Arithmetic Operator

Operator	Description	Example
+	Addition - Adds values on either side of the operator	A + B will give 30
-	Subtraction - Subtracts right hand operand from left hand operand	A - B will give -10
*	Multiplication - Multiplies values on either side of the operator	A * B will give 200
/	Division - Divides left hand operand by right hand operand	B / A will give 2
%	Modulus - Divides left hand operand by right hand operand and returns remainder	B % A will give 0
++	Increment - Increases the value of operand by 1	B++ gives 21
--	Decrement - Decreases the value of operand by 1	B-- gives 19

```
int quotient, remainder;  
quotient = 7 / 3;  
remainder = 7 % 3;  
System.out.println("quotient = " + quotient);  
System.out.println("remainder = " +  
remainder);
```

Example

```
public class Test{  
    public static void main(String[] args){  
        int a =10;  
        int b =20;  
        int c =25;  
        int d =25;  
  
        System.out.println("a + b = "+(a + b));  
        System.out.println("a - b = "+(a - b));  
        System.out.println("a * b = "+(a * b));  
        System.out.println("b / a = "+(b / a));  
        System.out.println("b % a = "+(b % a));  
        System.out.println("c % a = "+(c % a));  
        System.out.println("a++ = "+(a++));  
    }  
}
```

Example(cont'd)

```
System.out.println("a-- = "+(a--));  
// Check the difference in d++ and ++d  
System.out.println("d++ = "+(d++));  
System.out.println("++d = "+(++d));  
}  
}
```

■ This would produce the following result:

- $a + b = 30$
- $a - b = -10$
- $a * b = 200$
- $b / a = 2$
- $b \% a = 0$
- $c \% a = 5$
- $a++ = 10$
- $a-- = 11$
- $b-- = 11$
- $d++ = 25$
- $++d = 27$

The Relational Operator

- Assume variable A holds 10 and variable B holds 20

Operator	Description	Example
==	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
!=	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Example

```
public class Test{  
    public static void main(String[] args){  
        int a =10;  
        int b =20;  
        System.out.println("a == b = "+(a == b));  
        System.out.println("a != b = "+(a != b));  
    }  
}
```

■ This would produce the following result:

- a == b =false
- a != b =true

The Logical Operators

- Assume variable A holds 1 and variable B holds 0

Operator	Description	Example
&&	Called Logical AND operator. If both the operands are non-zero, then the condition becomes true.	(A && B) is false.
	Called Logical OR Operator. If any of the two operands are non-zero, then the condition becomes true.	(A B) is true.
!	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A && B) is true.

Example

```
public class Test{  
    public static void main(String[] args){  
        boolean t =true;  
        boolean f =false;  
        System.out.println("t && f = "+(t&&f));  
        System.out.println("t || f = "+(t||f));  
        System.out.println("!(t && f) = "+!(t && f));  
    }  
}
```

■ This would produce the following result:

- t && f =false
- t || f =true
- !(t && f)=true

The Assignment Operator

Operator	Description	Example
=	Simple assignment operator, Assigns values from right side operands to left side operand	$C = A + B$ will assign value of $A + B$ into C
+=	Add AND assignment operator, It adds right operand to the left operand and assign the result to left operand	$C += A$ is equivalent to $C = C + A$
-=	Subtract AND assignment operator, It subtracts right operand from the left operand and assign the result to left operand	$C -= A$ is equivalent to $C = C - A$
*=	Multiply AND assignment operator, It multiplies right	$C *= A$ is

Conditional Statements

■ Two types:

- If statements
- Switch statements

■ Syntax of if statements

```
if(expression){  
    //Statements will execute if the expression is true  
}
```

Example

```
public class Test {  
    public static void main(String[] args){  
        int x = 10;  
        if( x < 20 ){  
            System.out.print("This is if statement");  
        }  
    }  
}
```

Output

This is if statement

if...else statement

- An if statement can be followed by an optional else statement, which executes when the expression is false.

```
if(expression){  
    //Executes when the expression is true  
}else{  
    //Executes when the expression is false  
}
```


Example 1

```
public class Test2 {  
    public static void main(String[] args){  
        int x = 30;  
        if( x < 20 ){  
            System.out.print("This is if statement");  
        }  
  
        else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Output:

This is else statement

Example2

```
public class Test3 {  
    public static void main(String[] args){  
        int x = 30;  
        if( x == 10 ){  
            System.out.print("Value of X is 10");  
        }  
        else if( x == 20 ){  
            System.out.print("Value of X is 20");  
        }  
        else if( x == 30 ){  
            System.out.print("Value of X is 30");  
        }  
        else{  
            System.out.print("This is else statement");  
        }  
    }  
}
```

Activity

- Write a java program to check whether a number is even or not
- Write a java program that reads two integers and displays in ascending order.

```
import java.util.Scanner;
public class TwoSort{
    public static void main(String[] args){
        Scanner console = new Scanner(System.in);
        int a= console.nextInt();
        int b = console.nextInt();
        if (b < a){
            int t = a;
            a = b;
            b = t;
        }
        System.out.println(a);
        System.out.println(b);
    }
}
```

Exercise

- Write a java program that displays the following picture.

```
*****
 *               **
 *             * *
 *           * *
 *         * *
 *       * *
*****
*               *
*             *
*           *
*         *
*       *
*     *
*   *
* *
*
*****
```

- Write a java program that arrange three numbers in ascending order.

Switch

- Handles multiple conditions efficiently.

```
switch(expression){  
    case value :  
        //Statements  
        break; //optional  
    case value :  
        //Statements  
        break; //optional  
    //You can have any number of case statements.  
    default : //Optional  
        //Statements  
}
```

Example

```
public class SwitchTest {  
    public static void main(String[] args){  
        char grade = 'C';  
        switch(grade){  
            case 'A' :  
                System.out.println("Excellent!");  
                break;  
            case 'B' :  
                System.out.println("Well done");  
                break;  
            case 'C' :  
                System.out.println("You passed");  
                break;  
            case 'D' :  
                System.out.println("You Failed");  
            case 'F' :  
                System.out.println("Better try again");  
                break;  
            default :  
                System.out.println("Invalid grade");  
        }System.out.println("Your grade is " + grade);  
    }  
}
```

Loop

- There may be a situation when we need to execute a block of code several number of times, and is often referred to as a loop.
- You can use one of the following three loops:

- while Loop

```
while (condition) {  
    statements  
}
```

- for Loop

```
for (initialization; condition; update){  
    statements  
}
```

- do...while Loop

```
do {  
    (statements)  
} while ( boolean-expression );
```

Example 1

- Displaying all squares of numbers 1 to 10

```
public static void main(String[] args) {  
    int i=1;  
    while(i<=10){  
        System.out.println(i*i);  
        i++;  
    }  
}
```


Example 2: Nested loop

```
for (int i = 0; i < 3; i++) {  
    for (int j = 2; j < 4; j++) {  
        System.out.println (i + " " + j);  
    }  
}
```

Exercise

- Write a program to display sum of the first 100 integer numbers using for loop.
- Write statements that can be used in a Java program to read two integers and display the number of integers that lie between them, including the integers themselves. For example, four integers are between 3 and 6: 3, 4, 5, and 6.
- Write a program that reads an integer value and prints the sum of all even integers between 2 and the input value, inclusive. Print an error message if the input value is less than 2.
- Write a java program that accepts integer value from a user and display sum of each digit(E.g. input: 543, result: $5+4+3=12$)

Basic Classes

1. **System:** one of the core classes

- `System.out.println("Welcome to Java!");`
- `System.out.print("...");`
- `System.exit(0);`
- `System.out.println("50 plus 25 is " + 50 + 25);`

Basic Classes(cont'd)

2. Scanner:

- The **java.util.Scanner**: for accepting input from console
- Methods: `nextInt()`, `nextDouble()`, `nextLine()`, `next()`, ...
- E.g.:- `Scanner s = new Scanner (System.in);`
`int x = s.nextInt();`

Example 1

```
import java.util.Scanner; // so that I can use Scanner
public class Age{
    public static void main(String[] args) {
        Scanner console = new Scanner(System.in);
        System.out.print("How old are you? "); // prompt
        int age = console.nextInt();
        System.out.println("You'll be 30 in " + (30 - age) + " years.");
    }
}
```

Output:

How old are you? 18

You'll be 30 in 12 years

Example 2

```
import java.util.*;

public class HelloName{

    public static void main(String[] args){

        Scanner input= new Scanner(System.in);

        System.out.print("What is your name? ");

        String name = input.nextLine();

        System.out.println("Hello there " + name + ", nice to meet you!");

    }

}
```

Output:

What is your name? James

Hello there James, nice to meet you!

Example3

```
import java.util.Scanner; // program uses class Scanner
// Addition program that displays the sum of two numbers.
public class AdditionExample{
    // main method begins execution of Java application
    public static void main( String[] args){
        Scanner input = new Scanner( System.in );
        int number1; // first number to add
        int number2; // second number to add
        int sum; // sum of number1 and number2
        System.out.print( "Enter first integer: " ); // prompt
        number1 = input.nextInt(); // read first number from user
        System.out.print( "Enter second integer: " ); // prompt
        number2 = input.nextInt(); // read second number from user
        sum = number1 + number2; // add numbers
        System.out.println( "Sum is " + sum );
    } // end method main
} // end class Addition
```

Basic Classes(cont'd)

3. Math-Class

- The **java.lang.Math** class contains methods for performing basic numeric operations such as **exponential, logarithm, square root, and trigonometric functions**
 - `Math.ceil(x)` // `Math.ceil(3.2)` and `Math.ceil(3.9)` both return 4.0
 - `Math.floor(x)` // `Math.floor(3.2)` and `Math.floor(3.9)` both return 3.0
 - `Math.pow(x,y)` // `Math.pow(4,2)` returns 16.0
 - `Math.round(x)` // `Math.round(3.2)` returns 3; `Math.round(3.6)` returns 4
 - `Math.max(x,y)` // `Math.max(1, 2)` returns 2
 - `Math.sqrt(x)` // `Math.sqrt(4)` returns 2.0
 - `Math.random()`: a random floating point number between 0 and 1. generates a double value in the range [0,1)
 - Eg: `Math.random()*100`
 - `Math.toDegrees(x)`;
 - `Math.PI`
 - `Math.sin(x)`

Basic Classes(cont'd)

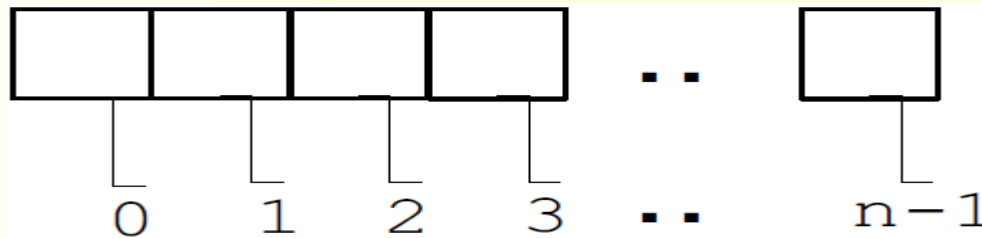
- Random is a class used to generate random numbers between the given lists/ranges.

- Example:

```
import java.util.Random;
public class RandomExample {
    public static void main(String[] args) {
        double x []=new double[20];
        Random r=new Random();
        Scanner s=new Scanner(System.in);
        for(int i=0;i<5;i++){
            x[i]=r.nextInt(100);
            System.out.println(x[i]);
        }
    }
}
```

Arrays

- Stores a fixed-size sequential collection of elements of the **same type**.
- The index starts at **zero** and ends at **length-1**.



- To create an array of a given size, use the operator new:
 - Data type arrayname[] = new dataType[size];
 - E.g. `int[] values = new int[5];`
- The for loop is ideally suited for performing array manipulations
 - E.g.

```
Int[] values = {9, 15, 18}; //initializing array
for (int i = 0; index < values.length; i++){
    System.out.println(values[i]);
}
```

Example

- Create an array with N random values

```
double[] a = new double[N];  
for (int i = 0; i < N; i++)  
    a[i] = Math.random();
```

- Copy to another

```
double[] b = new double[N];  
for (int i = 0; i < N; i++)  
    b[i] = a[i];
```

- Print array values, one per line

```
for (int i = 0; i < N; i++)  
    System.out.println(a[i]);
```

- Find the maximum of array values

```
double max = a[0];  
for (int i = 1; i < N; i++)  
    if (a[i] > max) max = a[i];
```

- Matrix addition

```
double[][] c = new double[N][N];  
for (int i = 0; i < N; i++)  
    for (int j = 0; j < N; j++)  
        c[i][j] = a[i][j] + b[i][j];
```

Cont'd

```
import java.util.*;
public class ArrayExample{
    public static void main(String[] args) {
        float x []=new float[5]; //array declaration
        System.out.println("enter any five numbers");
        for(int i=0;i<5;i++){
            x[i]=input.nextFloat();
            sum =sum+x[i];
        }
        System.out.println("result="+sum);
    }
}
```

Activity

- What is wrong with the following?

- `int[] temp = {7, 2, 1.5, 4};`

- `int[] temp= new int[5];`

- `temp[] = {7, 5, 9};`

- Write a program that prompts the user to enter two lists of integers and displays whether the two are identical

Exercise: Array

1. Write a java program that accepts 5 integers and display
 1. the smallest
 2. only evens
 3. Count only integers>50
2. Insert N-integers from the keyboard and calculate the sum and average
3. Generate 5 random numbers between 10 and 20 and find the smallest
4. Generate 5 random numbers between 1 and 50 and find the largest two
5. Insert a string using Scanner and count the number of 'a' in the string
6. Check if the inserted string is palindrome or not

Lab Exercise

- Write a program that reverses the digits of a given positive integer number entered from the keyboard(don't use array).
- Write a program that prompts the user to enter a string and displays the number of vowels and consonants in the string.
- Write a program that determines and print the number of odd, even, and zero digits in an integer value read from the keyboard
- Write a program that reads a string from the user and prints it one character per line.