CSA201 – Applied Data Structures and Algorithms

Unit 1: Introduction to Java Programming

What is Java?

- Java is computer programming language that enforces an **object-oriented programming paradigm**
- Java is a programming language and computing **platform** first released by **Sun Microsystems** in 1995
- Java was created by a team lead by **James Gosling**
- Java is a **platform independent programming language** that follows the logic of "**Write once, Run anywhere**"
- Java can be used to create complete applications that may run on a single computer or can be distributed among several servers and clients in a network

Java Features

- Simple
  - Java is easy to learn and its syntax is quite simple, clean and easy to understand.
- Robust
  - Java checks the code during the compilation time and run time
  - Java completely takes care of memory allocation and releasing, which makes Java more robust
- Secure
  - Java Programs runs inside virtual machine sandbox to prevent any activity from untrusted sources.
- High Performance
  - Although Java is an interpreted language. It was designed to support “Just-in-time” compilers, which dynamically compile bytecodes to machine code

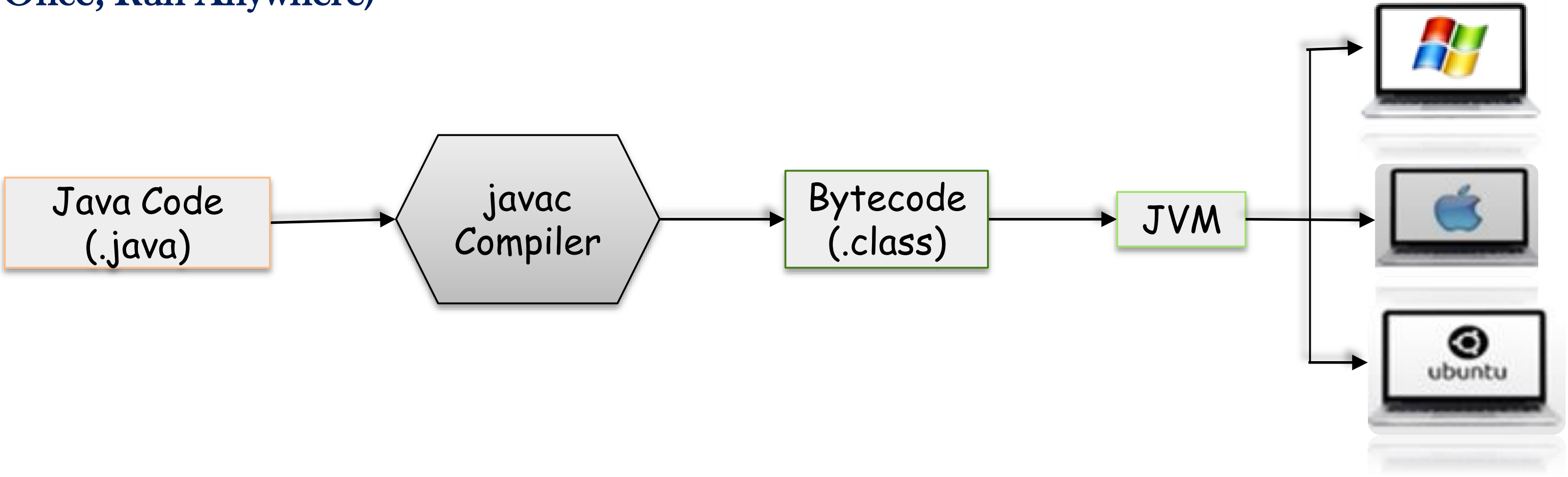
Java Features

- Portable
  - Applications written on one platform of Java can be easily ported to another platform as it is platform independent
- Distributed
  - RMI (Remote Method Invocation), EJB (Enterprise Java Beans) etc. are used for creating distributed applications using Java. Using this a program can call a method or another program running in some other computers in the network
- Multithreaded
  - Thread is a task in a process/program. Multithreading is multiple tasks running/executing at the same time
- Object-Oriented
  - Everything is performed using “objects”. Java can be easily extended since it is based on the Object model

Java Environment Setup

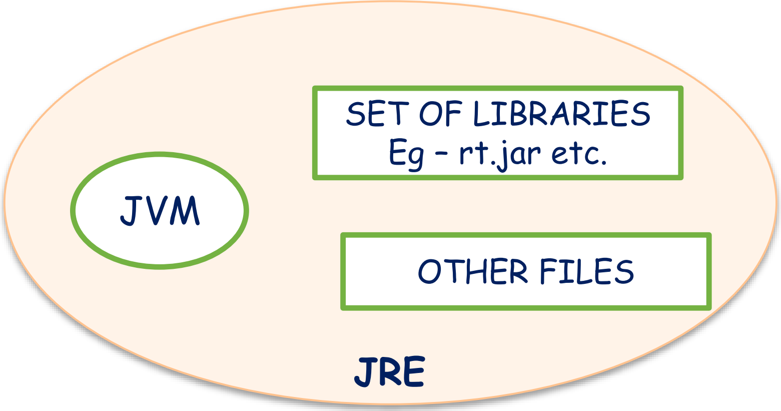
Java Virtual Machine (JVM)

- Java Virtual Machine (JVM) is the virtual machine that runs the Java **bytecodes**
- The JVM does not understand Java source code, that is why you compile your ***.java** files to obtain ***.class** files that contain the bytecodes understood by the JVM
- The same bytecodes give the same results makes Java a **Platform Independent Language (Write Once, Run Anywhere)**



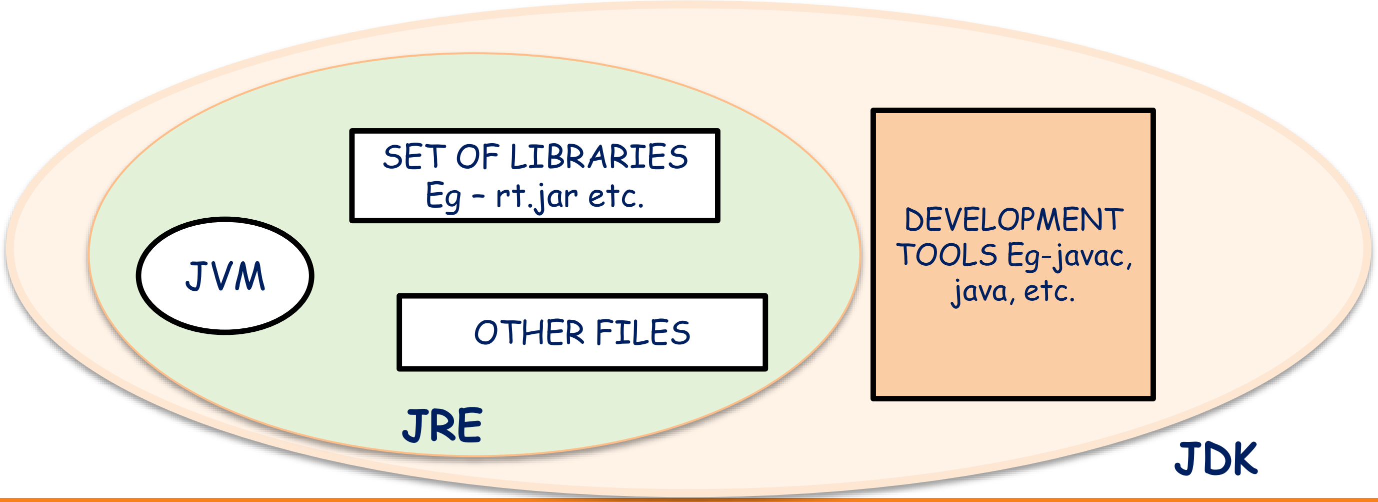
Java Runtime Environment (JRE)

- Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applications written in the Java programming language
  - **JRE = JVM + Set of libraries + Other Additional files**
- The JRE does not contain tools and utilities such as compilers or debuggers for developing application



Java Development Kit (JDK)

- JDK is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applications.
  - JDK = JRE + Development Tools
  - JDK = (JVM + Set of libraries + other additional files) + Development Tools

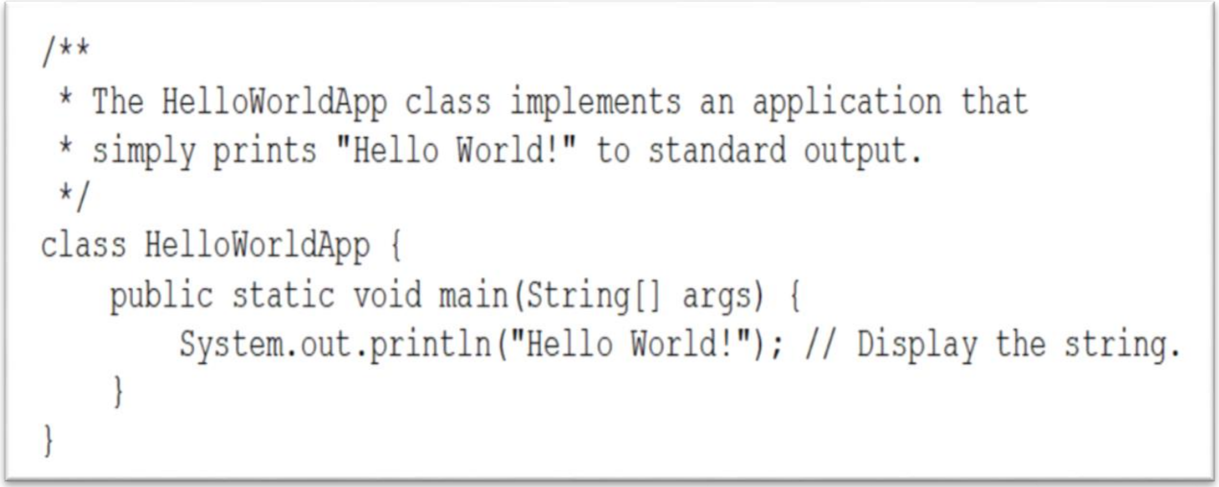

SET OF LIBRARIES Eg – rt.jar etc.
JVM
OTHER FILES
JRE
DEVELOPMENT TOOLS Eg-javac, java, etc.
JDK

A First Java Program

- To write your first program, you'll need:
  - The **Java SE Development Kit (JDK):** Download and install the JDK, which includes the JRE (and JVM)
    - For Microsoft Windows, Solaris OS, and Linux:
      - https://www.oracle.com/java/technologies/downloads/#java8-windows
  - The **VS Code** IDE
    - For all platforms:
      - https://code.visualstudio.com/download

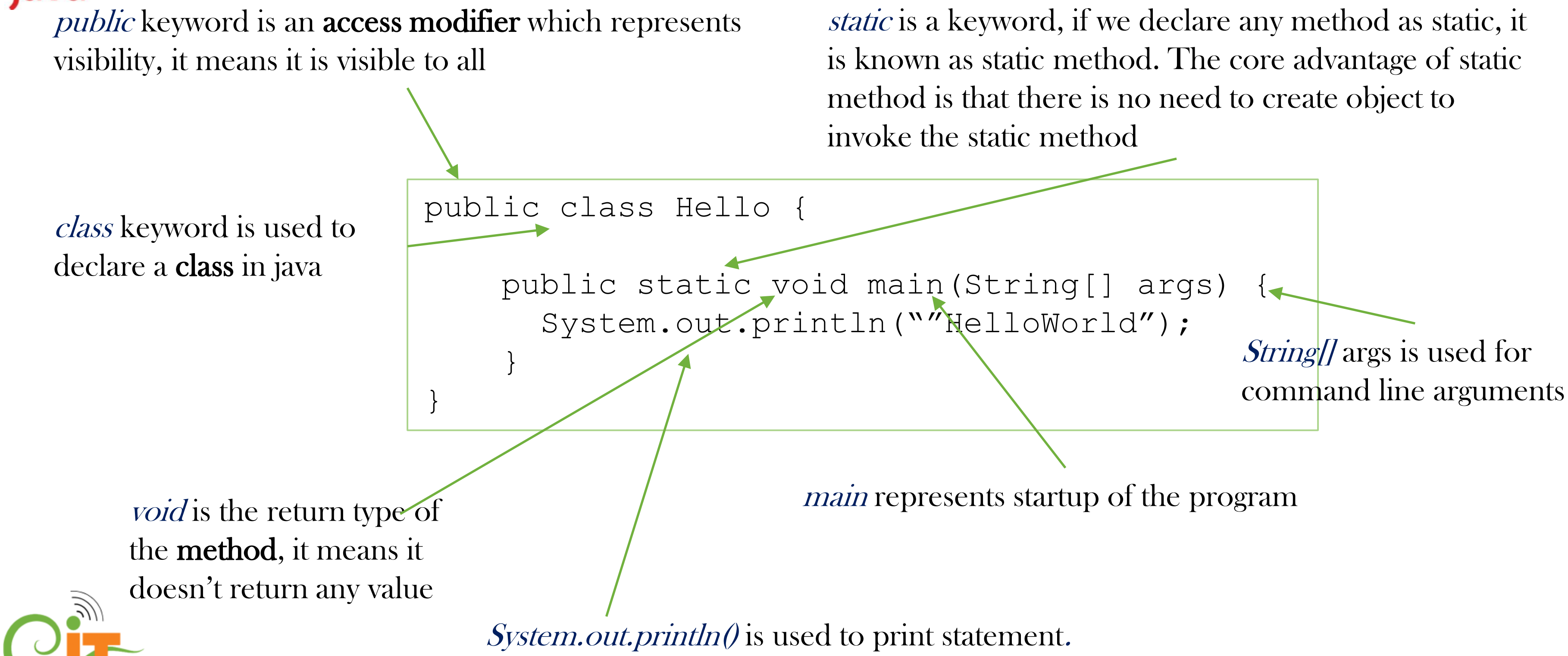
A First Java Program

- Your first application, **HelloWorldApp**, will simply display the greeting "Hello world!".
- *Step 1*: Create a source file
- A source file contains code, written in the Java programming language, that you and other programmers can understand. You can use **VS Code** to create and edit source files.



```
/**
 * The HelloWorldApp class implements an application that
 * simply prints "Hello World!" to standard output.
 */
class HelloWorldApp {
    public static void main(String[] args) {
        System.out.println("Hello World!"); // Display the string.
    }
}
```

A First Java Program



public keyword is an **access modifier** which represents visibility, it means it is visible to all

static is a keyword, if we declare any method as static, it is known as static method. The core advantage of static method is that there is no need to create object to invoke the static method

class keyword is used to declare a **class** in java

```
public class Hello {

    public static void main(String[] args) {
      System.out.println(“”HelloWorld”);
    }
}
```

String[] args is used for command line arguments

void is the return type of the **method**, it means it doesn’t return any value

main represents startup of the program

System.out.println() is used to print statement.

A First Simple Program

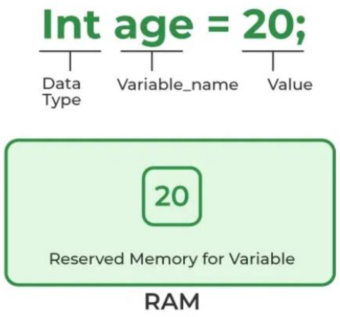
- *Step 2*: Compile the source file into a .class file
- The Java programming language compiler (**javac**) takes your source file and translates its text into instructions that the **Java Virtual Machine** can understand. The instructions contained within this file are known as **bytecodes.**

- *Step 3*: Run the program
- The Java application launcher tool (**java**) uses the **Java Virtual Machine** to run your application.

Basic Concepts of Java

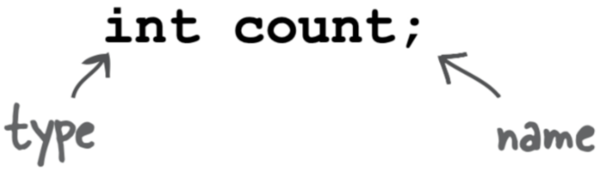
1. Variables

- What are Variables? Variables are memory locations which are reserved to store values
- Declaring Variables:
  - To use a variable, you need to declare it with a name and a data type.
  - Example: **int age;** declares an integer variable named "age."
- Assigning Values:
  - After declaring a variable, you can assign a value to it.
  - Example: **age = 20;** assigns the value 20 to the "age" variable.



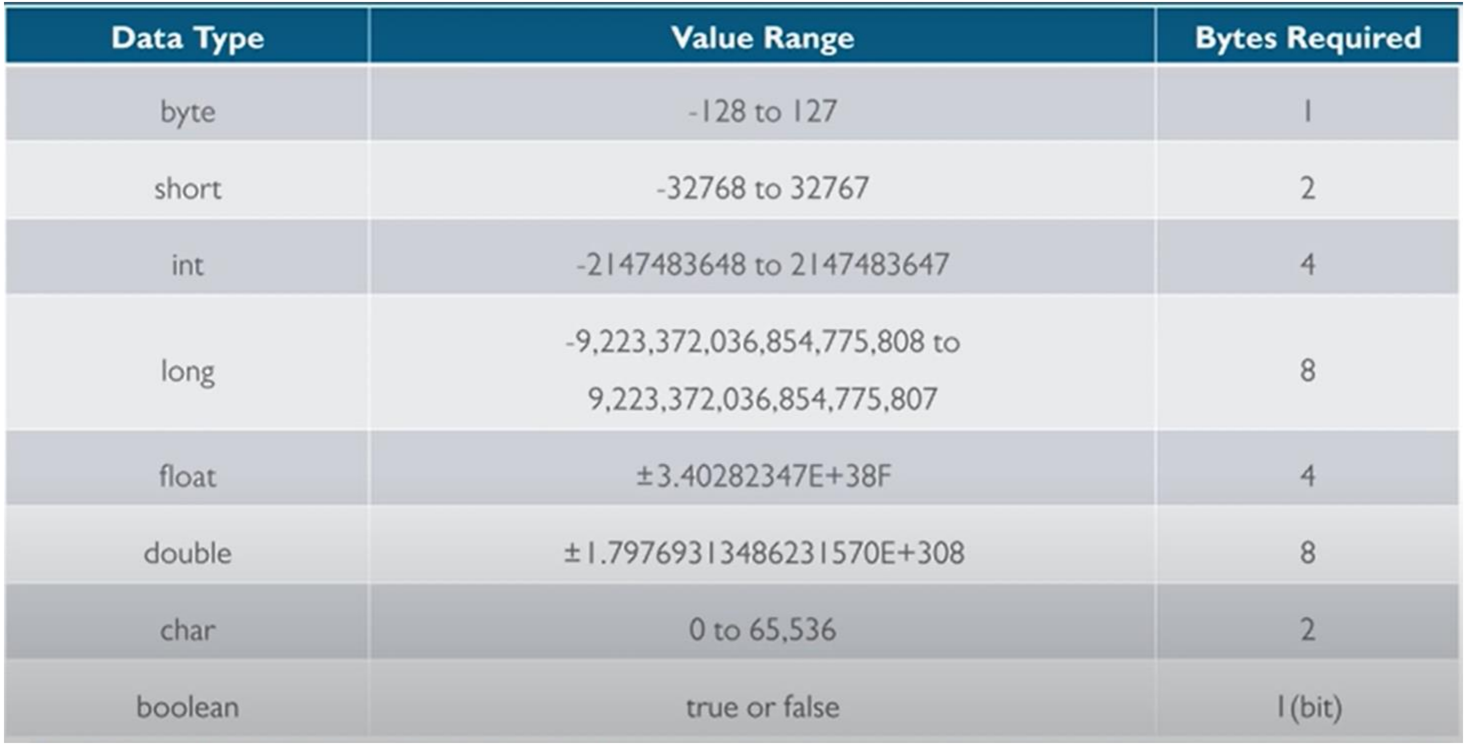
2. Datatypes

- Each variable in Java has a specific type, which **determines the size of memory, the range of values** that can be stored and **the set of operations** that can be applied to the variable



```
int count;
```

Primitive Datatypes



Data Type	Value Range	Bytes Required
byte	-128 to 127	1
short	-32768 to 32767	2
int	-2147483648 to 2147483647	4
long	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807	8
float	±3.40282347E+38F	4
double	±1.79769313486231570E+308	8
char	0 to 65,536	2
boolean	true or false	1(bit)

The equation used to find the range of values that can be stored in a variable is -2^(n-1) to +2^(n-1)-1 where “n” is number of bits

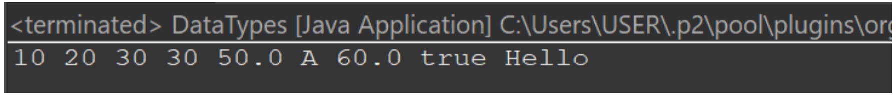
Demo – Variables and Data Types

```
public class DataTypes {

public static void main(String[] args) {
byte b = 10;
short s = 20;
int i = 30; long l = 40l;          1
float f = 50f;
char c = 'A';
double d = 60d;
boolean bin = true;
String str = "Hello";
System.out.println(b+" "+s+" "+i+" "+i+"
2 "+f+" "+c+" "+d+" "+bin+" "+str);

}}
```

Step 1: We will declare and initialize the variables with different datatypes

Step 2: Print all the variables

Output:

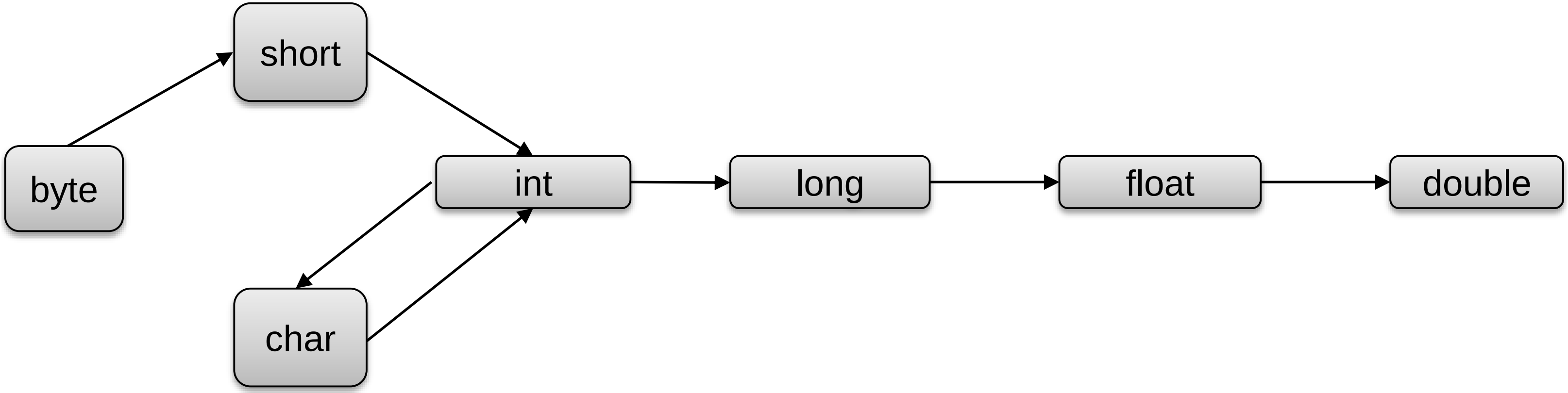


```
<terminated> DataTypes [Java Application] C:\Users\USER\.p2\pool\plugins\org
10 20 30 30 50.0 A 60.0 true Hello
```

Datatype Conversions

A datatype of a particular variable can be converted to other datatypes

- Sometimes necessary when working with different data types in operations or assignments.
- There are two ways in which we can perform Datatype Conversion:
  1. **Implicit Conversion:** Java performs implicit conversions automatically when no data loss is expected. For example, converting an int to a double.



Datatype Conversions

2. **Explicit conversion (casting)**:
   - When data loss might occur, explicit conversion (casting) is needed.
   - Use **(datatype)** before a value to indicate the desired data type.
   - Example: (int) 3.14 converts a double to an int.
   - Types:
     a) Widening Conversion
     b) Narrowing Conversion

Datatype Conversions

- Widening and Narrowing Conversion:
  - Widening (Upcasting): Converting to a larger data type without data loss.
  - Narrowing (Downcasting): Converting to a smaller data type, which can result in data loss.

  - *Example:* Widening Conversion:

    int to double is a widening conversion.

    ```
    Example: int num = 10;
             double result = num;
    ```

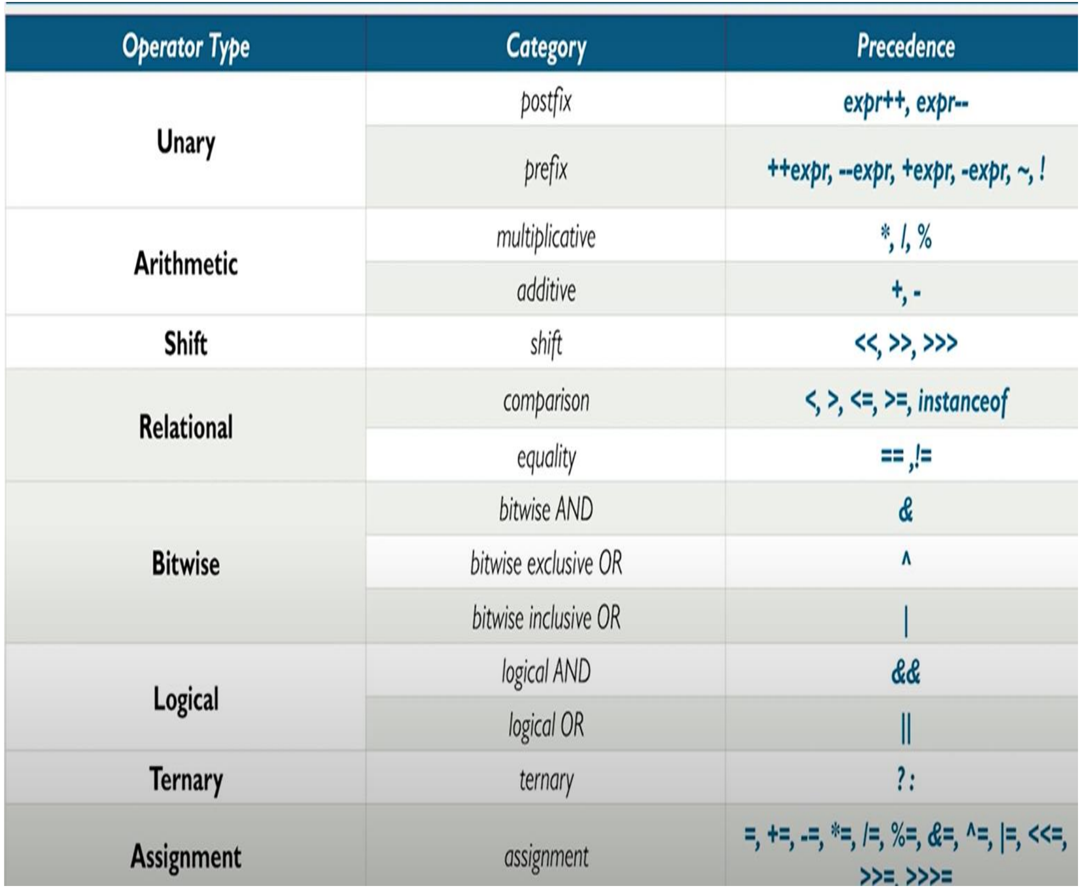
  - *Example:* Narrowing Conversion:

    double to int requires explicit casting.

    ```
    Example: double num = 5.75;
             int intValue = (int) num;
    ```

3. Operators

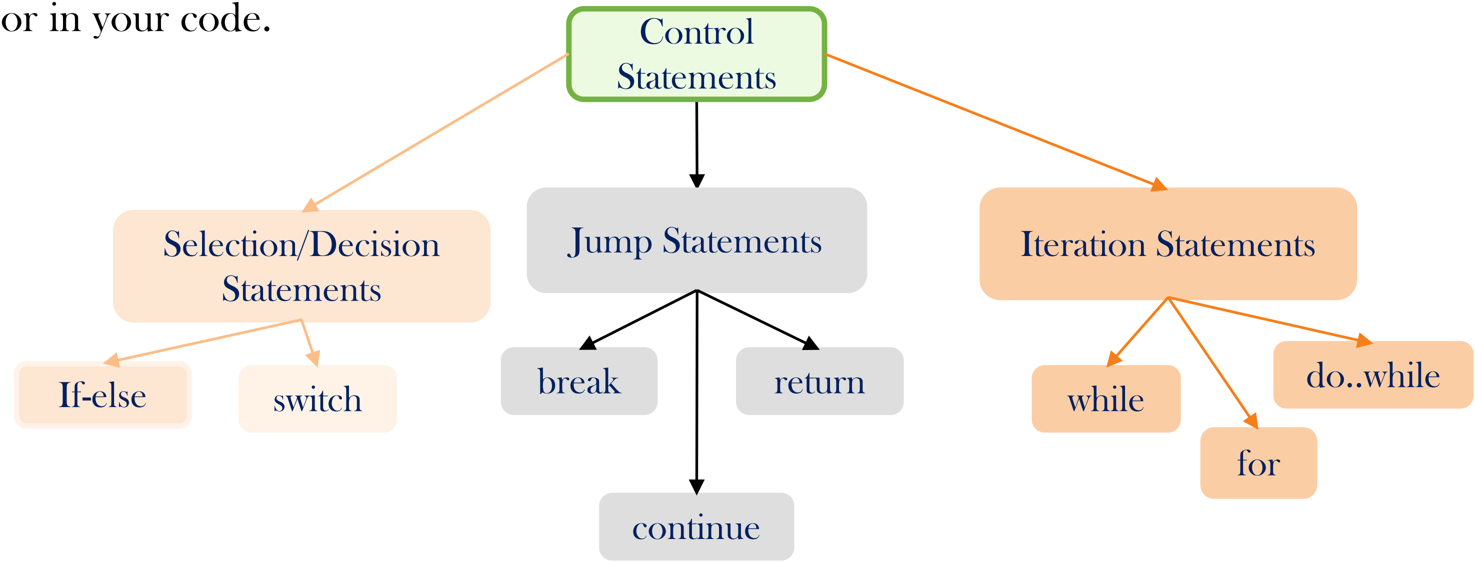
- **Operators** in Java are symbols that represent operations performed on variables and values.
- They enable programmers to manipulate data, compare values, and perform calculations in Java programs.



Operator Type	Category	Precedence
Unary	postfix	expr++, expr--
	prefix	++expr, --expr, +expr, -expr, ~, !
Arithmetic	multiplicative	*, /, %
	additive	+, -
Shift	shift	<<, >>, >>>
Relational	comparison	<, >, <=, >=, instanceof
	equality	== ,!=
Bitwise	bitwise AND	&
	bitwise exclusive OR	^
	bitwise inclusive OR	|
Logical	logical AND	&&
	logical OR	||
Ternary	ternary	? :
Assignment	assignment	=, +=, -=, *=, /=, %=, &=, ^=, |=, <<=, >>=, >>>=

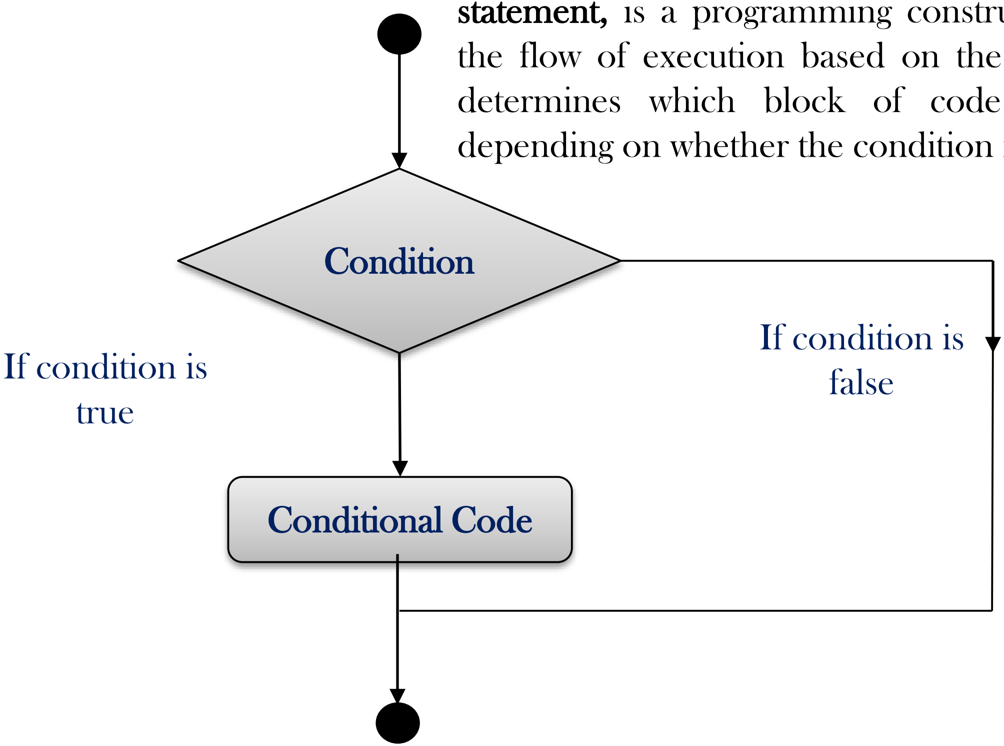
4. Control Statements

Control statements are fundamental structures in programming that enable you to control the flow of your program's execution. They allow you to make decisions, repeat actions, and create dynamic behavior in your code.



Selection Statements – Flow Diagram

A **selection statement, also known as a conditional/decisional statement,** is a programming construct that allows you to control the flow of execution based on the evaluation of a condition. It determines which block of code should be executed next, depending on whether the condition is true or false.

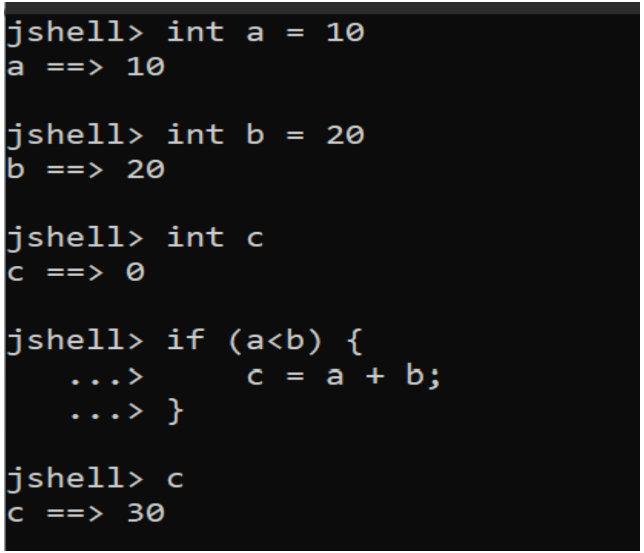


Selection Statements - *if* Statement

- An if statement consists of a expression, which is checked and returns a Boolean value, if true will execute the statements in the if block
- Syntax:

```
if (condition)
  {
  //statements
  }
```

Example:



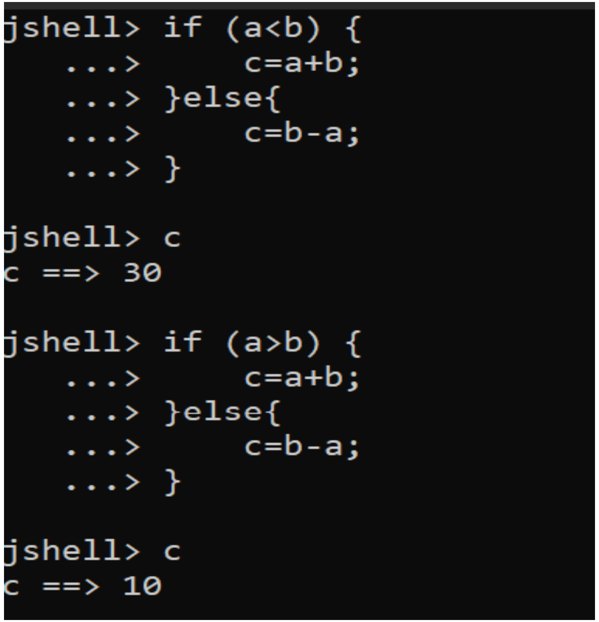
```
jshell> int a = 10
a ==> 10

jshell> int b = 20
b ==> 20

jshell> int c
c ==> 0

jshell> if (a<b) {
   ...>     c = a + b;
   ...> }

jshell> c
c ==> 30
```

Selection Statements - *if-else* Statement

- An if statement can be followed by an optional else statement, which executes when the if condition is false
- Syntax:

```
if (condition){
//statements
}else {
//statements
}
```

Example:



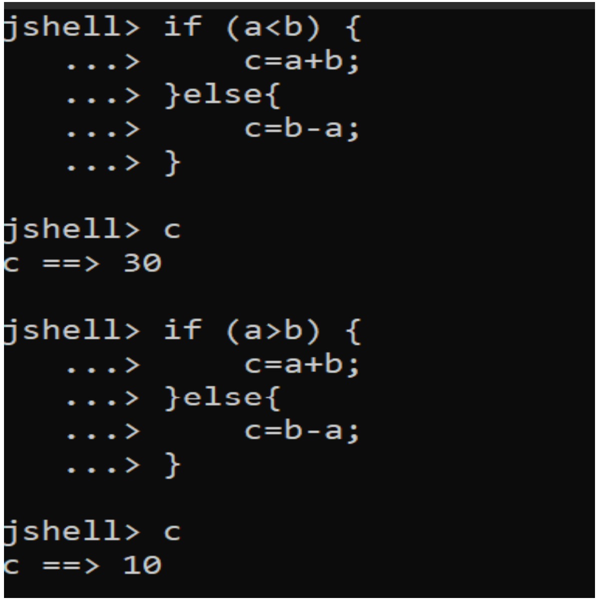
```
jshell> if (a<b) {
   ...>       c=a+b;
   ...> }else{
   ...>       c=b-a;
   ...> }

jshell> c
c ==> 30

jshell> if (a>b) {
   ...>       c=a+b;
   ...> }else{
   ...>       c=b-a;
   ...> }

jshell> c
c ==> 10
```

Selection Statements - Nested *if-else* Statement

- An if statement consists of a expression, which is checked and returns a Boolean value, if true will execute the statements in the if block
- Syntax:

```
if (condition1){
//statements
}else {
//statements
}
if (condition2) {
//statements
}
```

Example:



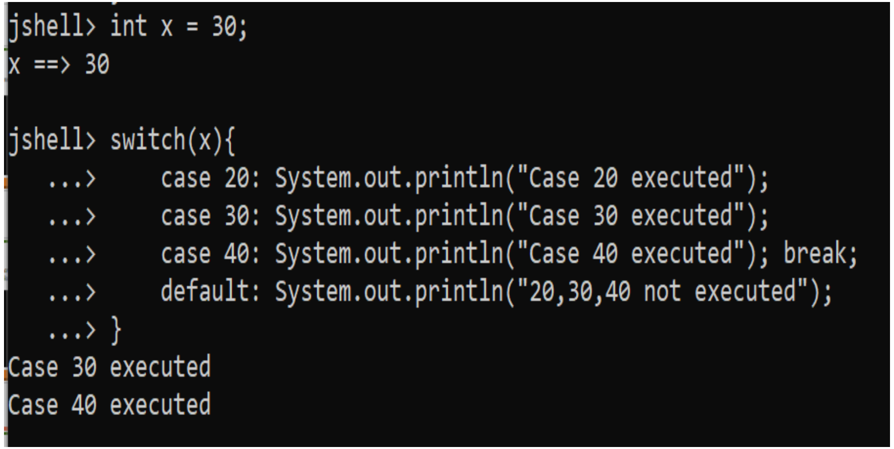
```
jshell> if (a<b) {
   ...>        c=a+b;
   ...> }else{
   ...>        c=b-a;
   ...> }

jshell> c
c ==> 30

jshell> if (a>b) {
   ...>        c=a+b;
   ...> }else{
   ...>        c=b-a;
   ...> }

jshell> c
c ==> 10
```

Selection Statements - *switch* Statement

- A switch statement allows a variable to be tested for equality against a list of values
- Syntax:

```
switch (expression){
case value1: //code to be executed
             break; //optional
case value2: //code to be executed
             break; //optional
default:  //code to be executed if all
cases are not matched
}
```

Example:

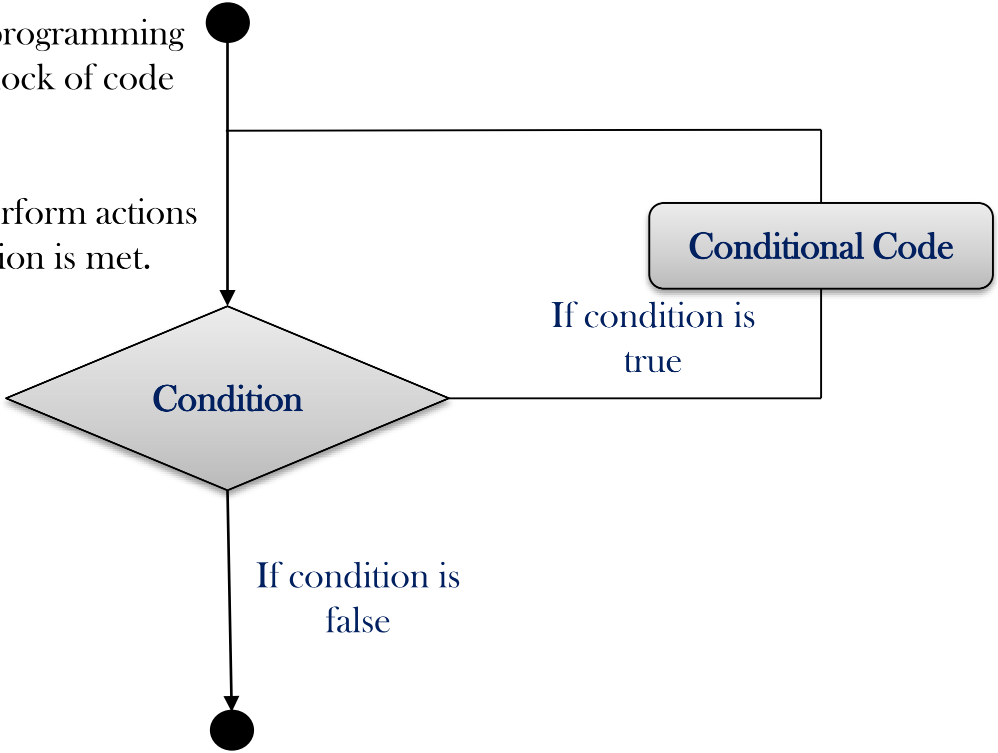


```
jshell> int x = 30;
x ==> 30

jshell> switch(x){
   ...>     case 20: System.out.println("Case 20 executed");
   ...>     case 30: System.out.println("Case 30 executed");
   ...>     case 40: System.out.println("Case 40 executed"); break;
   ...>     default: System.out.println("20,30,40 not executed");
   ...> }
Case 30 executed
Case 40 executed
```

Iteration Statements – Flow Diagram

An iteration statement, also known as a loop, is a programming construct that allows you to repeatedly execute a block of code as long as a certain condition is satisfied.

Loops are used to automate repetitive tasks and perform actions a specific number of times or until a certain condition is met.

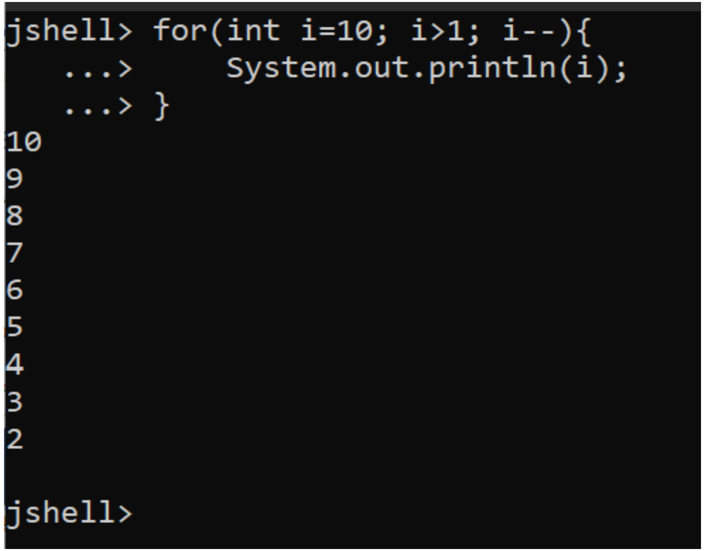


Iteration Statements - Simple *for* loop

- We can initialize variable, check condition (repeat if true) and increment/decrement value
- Syntax:

```
for (initialization; condition;
increment/decrement){
  //code to be executed
}
```

Example:



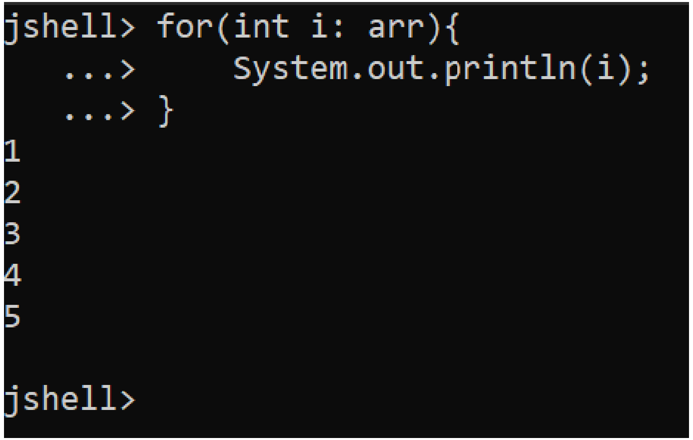
```
jshell> for(int i=10; i>1; i--){
   ...>     System.out.println(i);
   ...> }
10
9
8
7
6
5
4
3
2

jshell>
```

Iteration Statements - *for-each* loop

- It is used to traverse array or collection in Java
- It is easier to use than simple for loop because we don’t need to increment value and use subscript notation
- Syntax:

```
for (Type var : array){
  //code to be executed
}
```

Example:



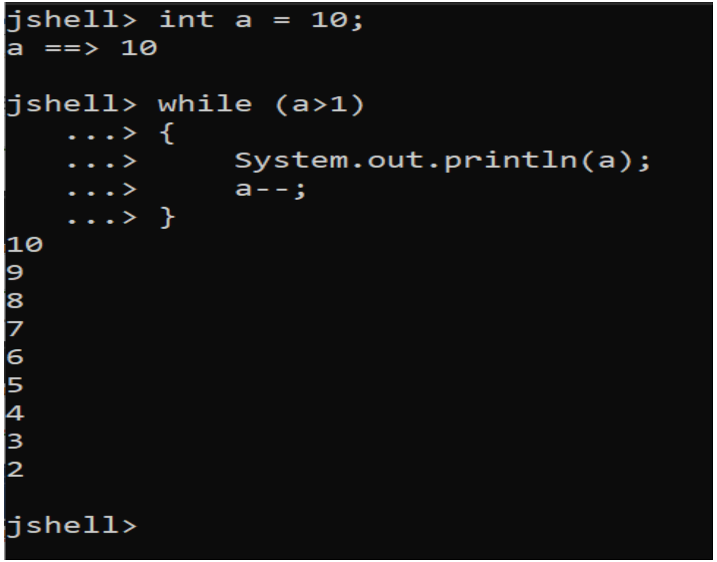
```
jshell> for(int i: arr){
   ...>     System.out.println(i);
   ...> }
1
2
3
4
5

jshell>
```

Iteration Statements - Simple *while* loop

- While loop is used to iterate a part of the program several times when the number of iterations is not known
- The loop keeps on iterating till the condition returns a False value
- Syntax:

```
while(condition){
  //code to be executed
}
```

Example:



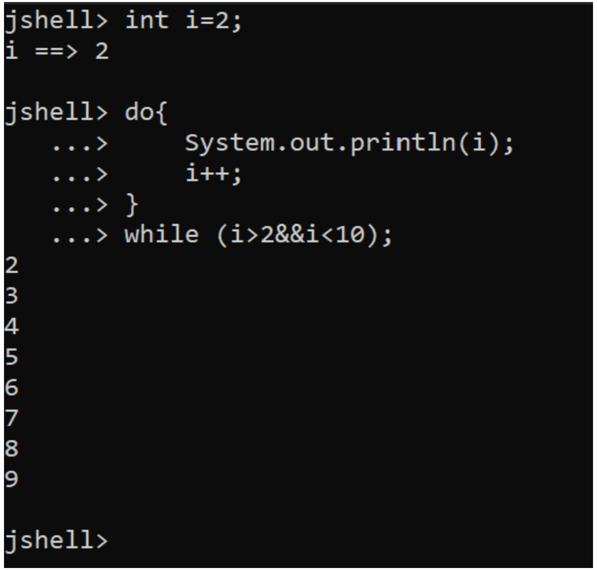
```
jshell> int a = 10;
a ==> 10

jshell> while (a>1)
   ...> {
   ...>     System.out.println(a);
   ...>     a--;
   ...> }
10
9
8
7
6
5
4
3
2

jshell>
```

Iteration Statements - *do…while* loop

- If the number of iteration is not fixed and you must have to execute the loop at least once, a do-while is used
- The do-while loop is executed at least once because condition is checked after loop body
- Syntax:

```
do{
  //code to be executed
} while(condition);
```

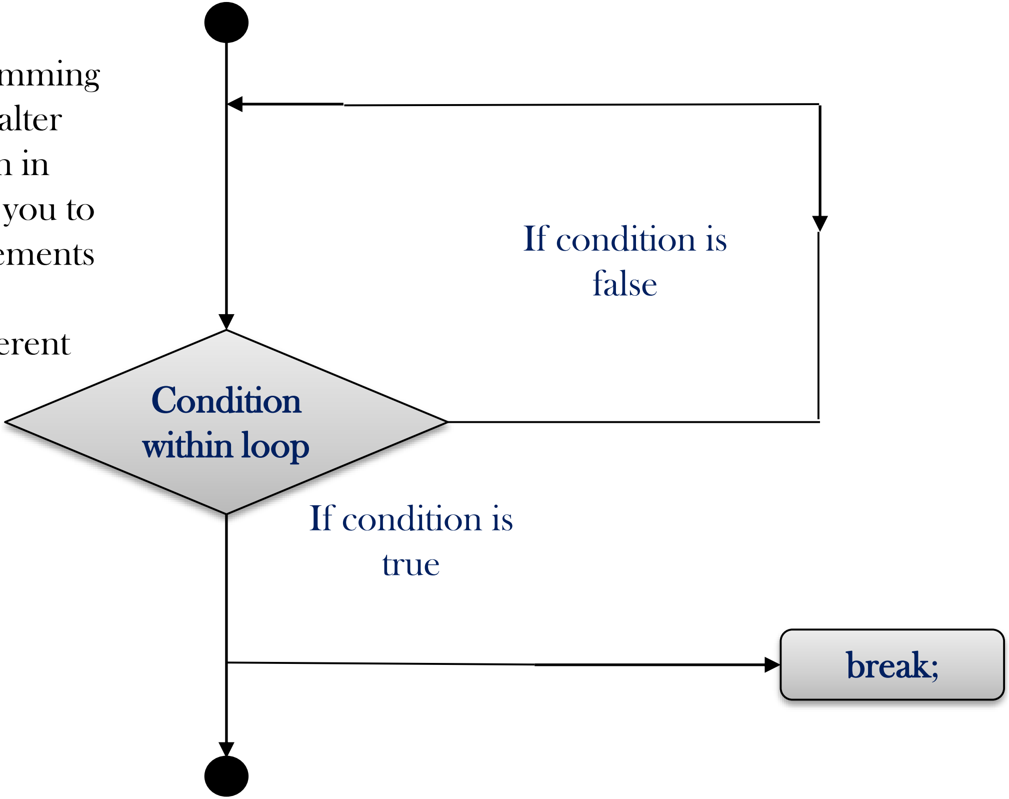
Example:



```
jshell> int i=2;
i ==> 2

jshell> do{
   ...>     System.out.println(i);
   ...>     i++;
   ...> }
   ...> while (i>2&&i<10);
2
3
4
5
6
7
8
9

jshell>
```

Jump Statements – Flow Diagram

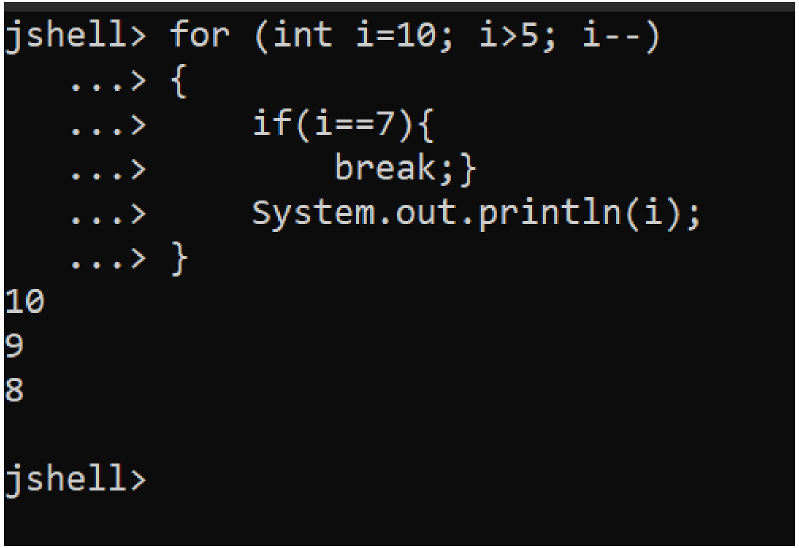
Jump statements are programming constructs that allow you to alter the normal flow of execution in your program. They enable you to change the sequence of statements being executed, often by transferring control to a different part of the code.



Jump Statements - *break* statement

- The break statement is used to break loop or switch statement. It breaks the current flow of the program at specified condition.

Example:

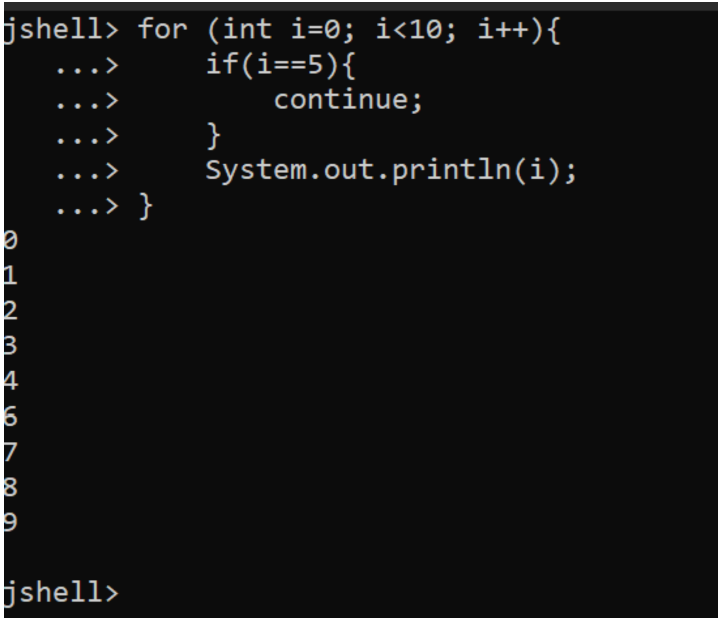


```
jshell> for (int i=10; i>5; i--)
   ...> {
   ...>     if(i==7){
   ...>         break;}
   ...>     System.out.println(i);
   ...> }
10
9
8

jshell>
```

Jump Statements - *continue* statement

- The continue statement is used to continue loop. It continues the current flow of the program and skips the remaining code at specified condition

Example:



```
jshell> for (int i=0; i<10; i++){
   ...>     if(i==5){
   ...>         continue;
   ...>     }
   ...>     System.out.println(i);
   ...> }
0
1
2
3
4
6
7
8
9

jshell>
```

Thank You ☺