```
System.out.println("Hi!");
```

```
main() method signature
```

Java
CHEAT SHEET

SCALER Topics

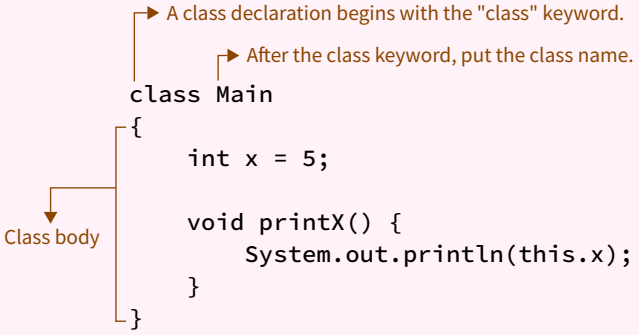
```
nextBoolean()
```

```
byte a = 13;
```

Basic Syntax

Class Declaration

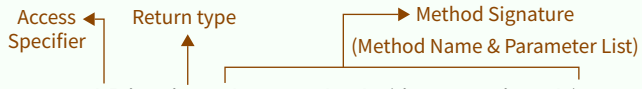
Create a class with name: Main




```
class Main
{
    int x = 5;

    void printX() {
        System.out.println(this.x);
    }
}
```

Method Declaration

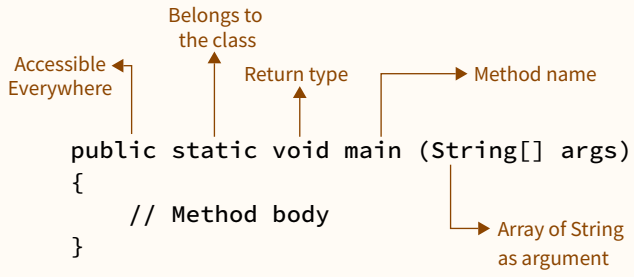
Create a method with name: demoMethod




```
public int demoMethod (int a, int b)
{
    // Method body
}
```

main() method signature

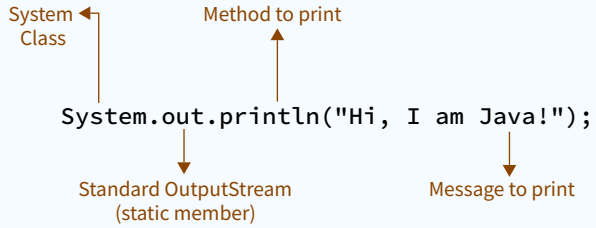
Create a method with name: demoMethod




```
public static void main (String[] args)
{
    // Method body
}
```

System output

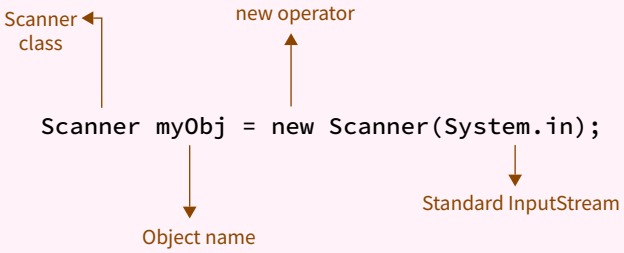
Create a method with name: demoMethod




```
System.out.println("Hi, I am Java!");
```

System input

Create an object of Scanner class: myObj




```
Scanner myObj = new Scanner(System.in);
```

Use one of these methods to take input:

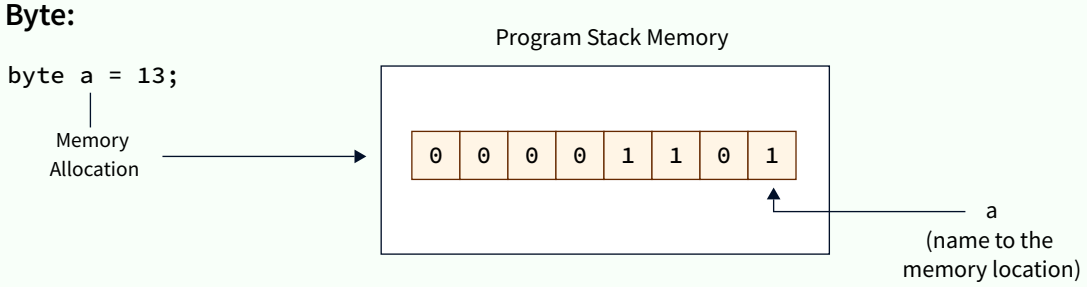
`nextBoolean()`	- Reads a boolean value from the user
`nextByte()`	- Reads a byte value from the user
`nextDouble()`	- Reads a double value from the user
`nextFloat()`	- Reads a float value from the user
`nextInt()`	- Reads an int value from the user
`nextLine()`	- Reads a String value from the user
`nextLong()`	- Reads a long value from the user
`nextShort()`	- Reads a short value from the user

```
String username = myObj.nextLine();
// Take the whole line as string input
```

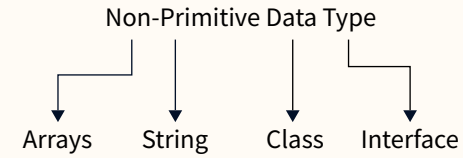
Primitive Data Types

Type	Size	Range	Default Values
byte	8 bits	-128 to 127	0
short	16 bits	-32,768 to 32,767	0
int	32 bits	-2^31 to 2^31 - 1	0
long	64 bits	-2^63 to 2^63 - 1	0L
float	32 bits	1.4E-45 to 3.4028235E38	0.0f
double	64 bits	4.9E-324 to 1.797693E308	0.0d
char	16 bits	'\u0000' to '\uffff'	'\u0000'
boolean	1 bit	true or false	false

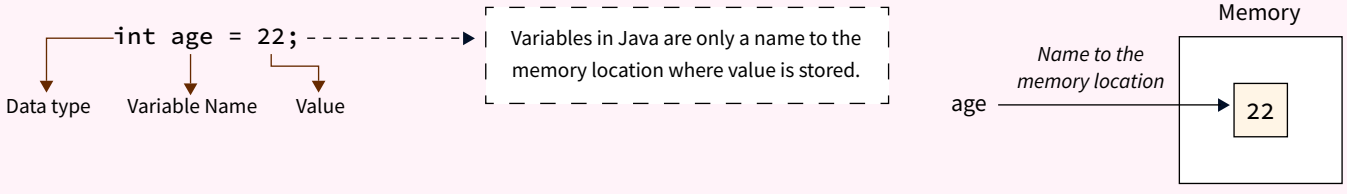
Byte:



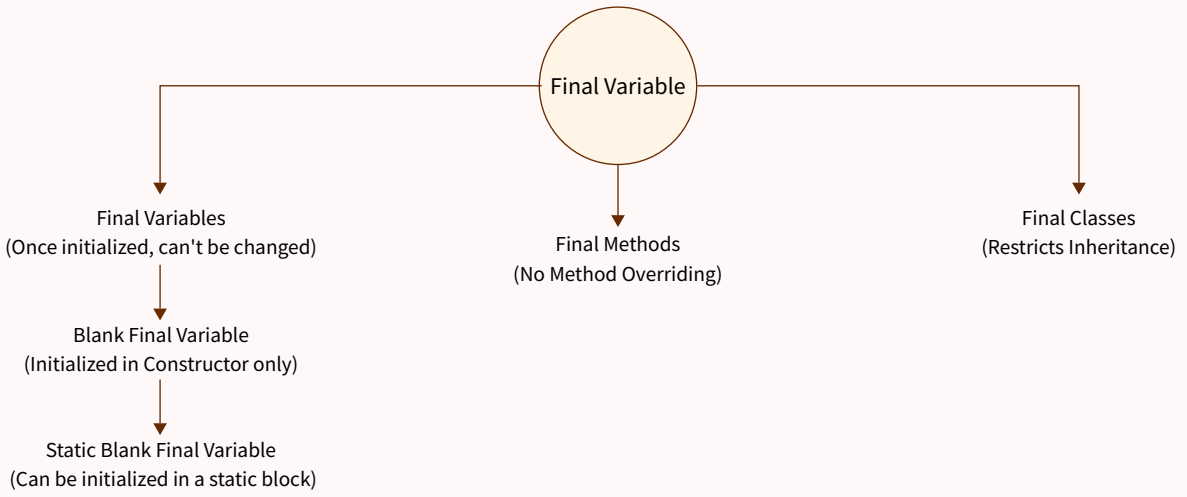
Non-Primitive Data Types



Java Variables



Final Keyword in Java



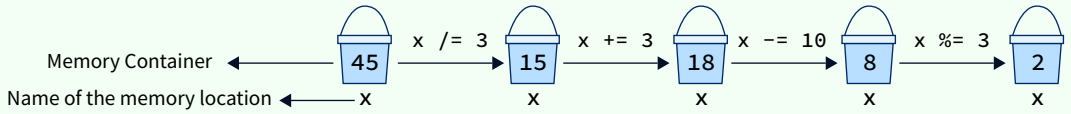
Operators

Arithmetic Operators

```
System.out.println(5 + 3);  // Addition, Output: 8
System.out.println(5 - 3);  // Subtraction, Output: 2
System.out.println(5 * 3);  // Multiplication, Output: 15
System.out.println(5 / 3);  // Division, Output: 1.6666666666666667
System.out.println(5 % 3);  //Modulo, Output: 2
```

Assignment Operators

```
int x = 45;                 // Assignment operator
x /= 3;                     // Equivalent to x = x / 3;
x += 3;                     // Equivalent to x = x + 3;
x -= 10;                    // Equivalent to x = x - 10;
x %= 3;                     // Equivalent to x = x % 3;
System.out.println(x);      // Output: 2
```

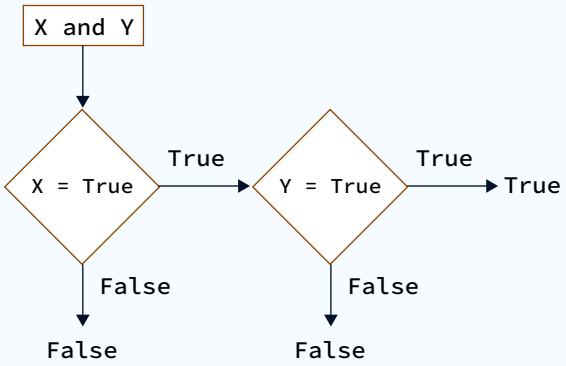


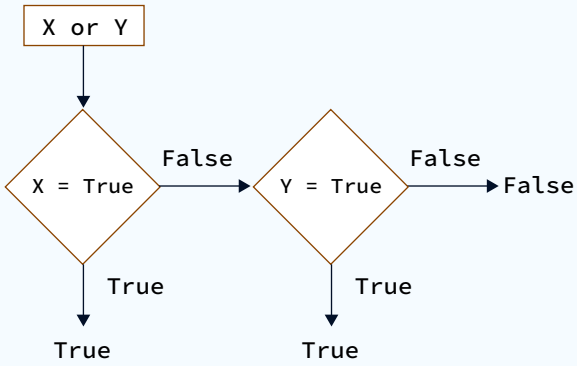
Comparison Operators

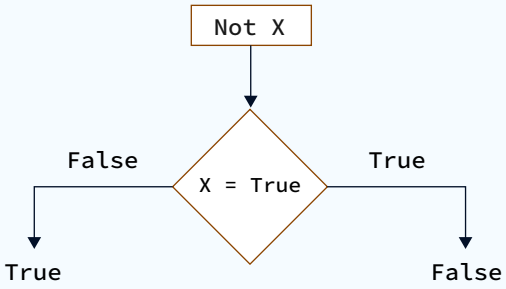
```
System.out.println(5 == 3);  // Equal, Output: false
System.out.println(5 != 3);  // Not equal, Output: true
System.out.println(5 > 3);   // Greater than, Output: true
System.out.println(5 < 3);   // Less than, Output: false
System.out.println(5 >= 3);  // Greater than or equal to, Output: true
System.out.println(5 <= 3);  // Less than or equal to, Output: false
```

Logical Operators (and/or/not)

```
System.out.println(true && false);      // Logical AND, Output: false
System.out.println(true || false);      // Logical OR, Output: true
System.out.println(!true);              // Logical NOT, Output: false
```

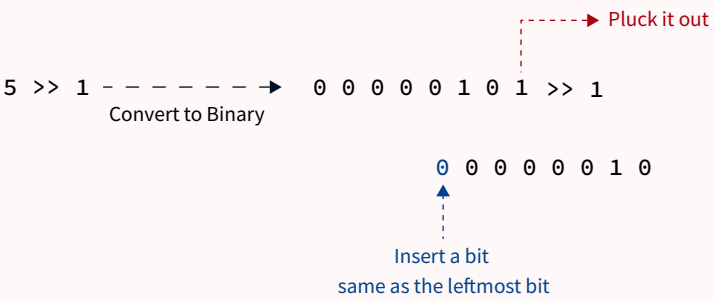






Bitwise Operators

```
System.out.println(5 & 3); // Bitwise AND, Output: 1
System.out.println(5 | 3); // Bitwise OR, Output: 7
System.out.println(5 ^ 3); // Bitwise XOR, Output: 6

System.out.println(~5); // Bitwise NOT, Output: -6
System.out.println(5 >> 1); // Bitwise Right Shift, Output: 2
System.out.println(5 << 1); // Bitwise Left Shift, Output: 10
```



Control Structures

if or if-else statements

```
int age = 22;
if (age < 18) {  [Condition False]
    System.out.println("Teenager!");
}
else {
    System.out.println("Adult!");
}
```

if or else-if or if-else statements

```
int age = 22;
if (age < 12) {  [Condition False]
    System.out.println("Child");
}
else if (age < 18) {  [Condition False]
    System.out.println("Teenager");
}
else if (age < 40) {  [Condition True]
    System.out.println("Adult");
}
else {  [Ignored]
    System.out.println("Old age");
}
System.out.println("End");
```

nested if statements

```
int age = 22;
if (age < 18) {  [Condition False]
    if (age < 12) {
        System.out.println("Child");
    }
    else {
        System.out.println("Teenager");
    }
}
else {
    if (age < 40) {  [Condition True]
        System.out.println("Adult");
    }
    else {
        System.out.println("Old Age");
    }
}
```

switch-case statement

```
char op = '*';
int n1 = 5, n2 = 7;
switch(op) {
    case '+':   [Not Matched]
        printf("n1 + n2 = %d",n1+n2);
        break;
    case '-':   [Not Matched]
        printf("n1 - n2 = %d",n1-n2);
        break;
    case '*':
        printf("n1 * n2 = %d",n1*n2);
        break;
    case '/':   [Ignored]
        printf("n1 / n2 = %d",n1/n2);
        break;
    // operator doesn't match any case constant +, -, *, /
    default:    [Ignored]
        printf("Error! operator is not correct");
}
```

for statement

Signature:

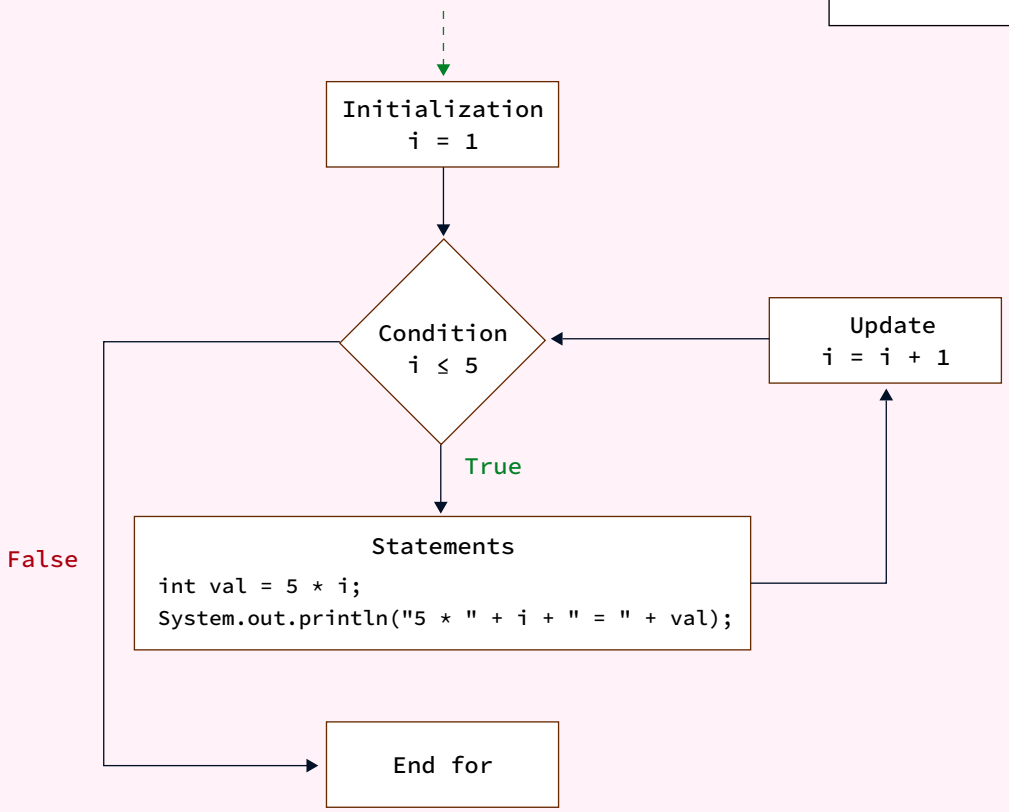
```
for (initialization; test; update) {
    STATEMENT(s);
}
```

Example:

```
for(int i = 1; i <= 5; i = i + 1) {
       int val = 5 * i;
       System.out.println("5 * " + i + " = " + val);
}
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```



while statement

Signature:

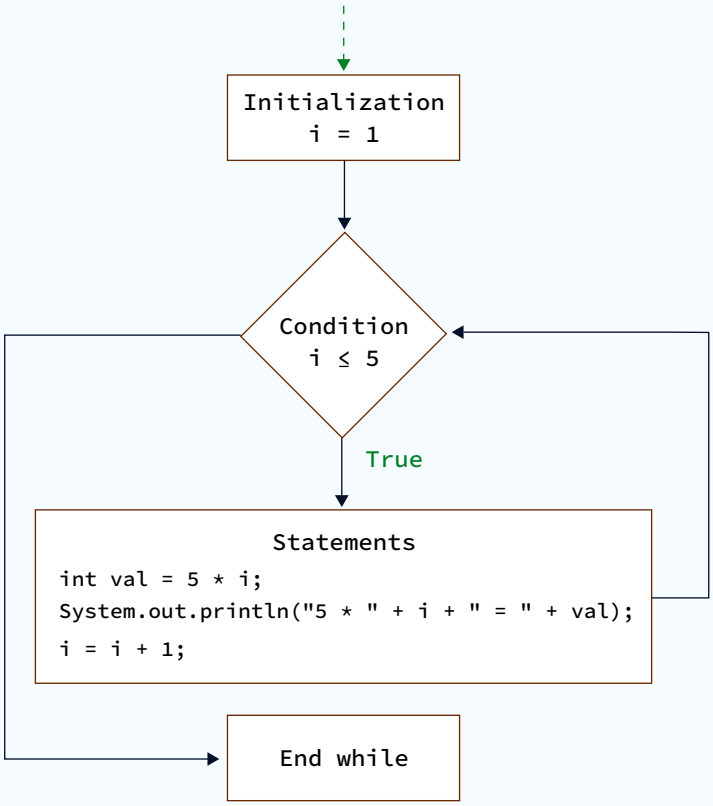
```
while(test) {
    STATEMENT(s);
}
```

Example:

```
int i = 1;
while (i <= 5) {
    int val = 5 * i;
    System.out.println("5 * " + i + " = " + val);

    i = i + 1;
}
```

Output

```
5 * 1 = 5
5 * 2 = 10
5 * 3 = 15
5 * 4 = 20
5 * 5 = 25
```



do-while statement

Signature:

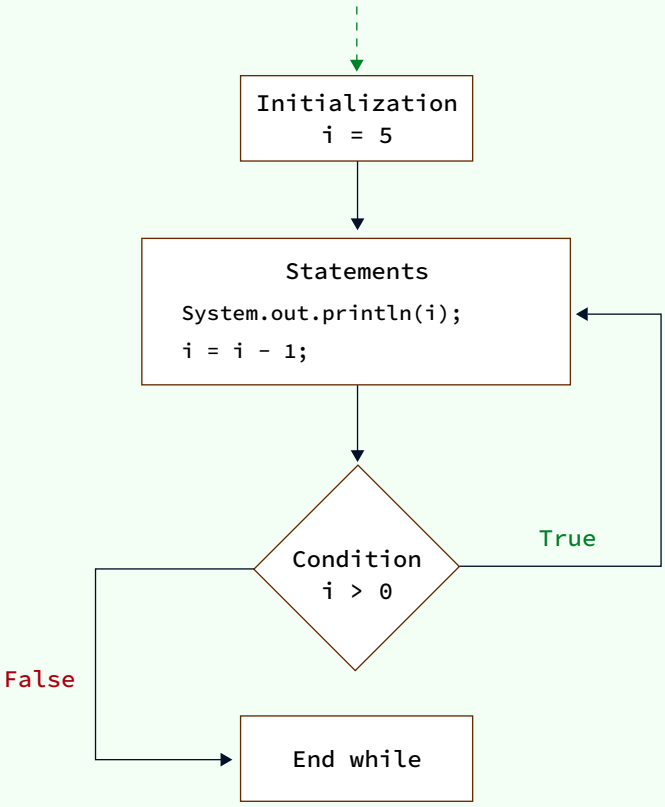
```
do {
    STATEMENT(s);

} while(test);
```

Example:

```
int i = 5;
do{
   System.out.println(i);
   i = i - 1;

} while(i > 0);
```

Output

```
5
4
3
2
1
```

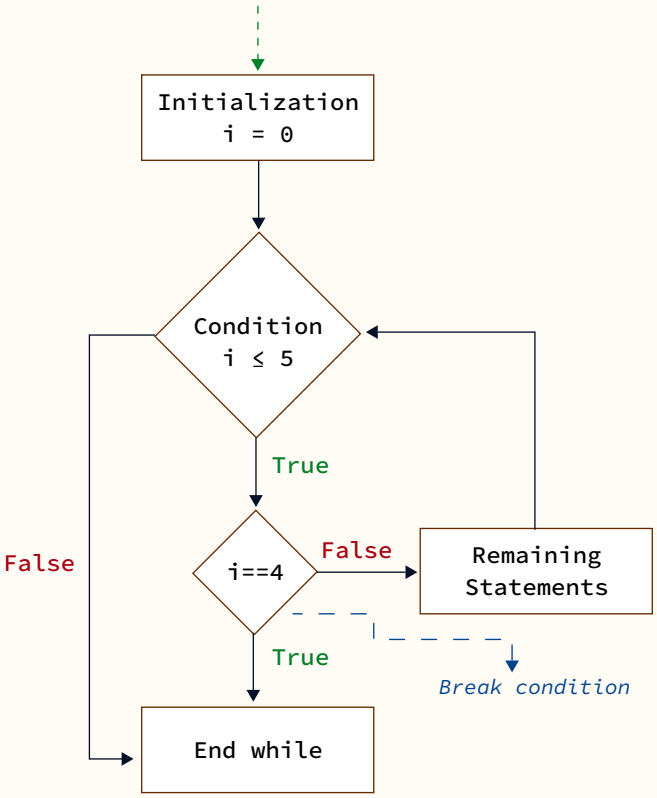


Break and Continue

```
int i = 0;
while (i <= 5) {
    if (i == 4) {
        break;
    }

    System.out.println(i);
    i = i + 1;
}
```

Output

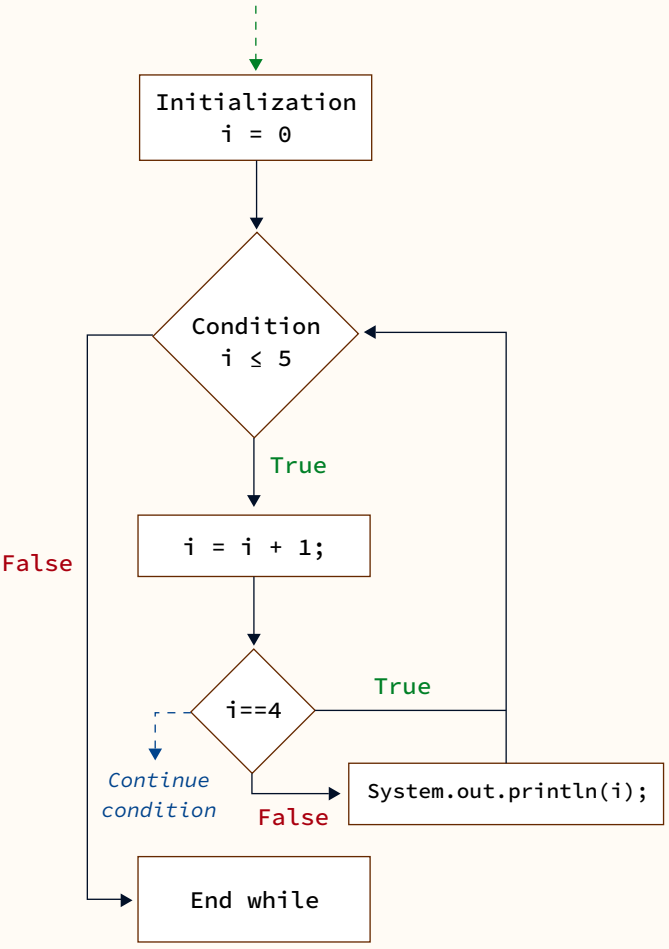
```
0
1
2
3
```




```
int i = 0;
while (i <= 5) {
    i = i + 1;

    if (i == 4) {
        continue;
    }
    System.out.println(i);
}
```

Output

```
1
2
3
5
6
```



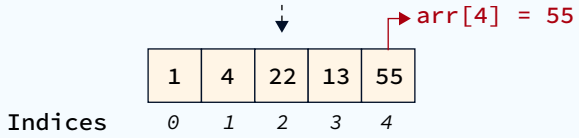
Arrays

Declaration:

Method 1:
```
int[] arr = new int[5];
```

Method 2:
```
int[] arr = new int[]{3, 5, 1, 2, 3};
```

```
int[] arr = new int[]{1, 4, 21, 13, 55};
```



Four Pillars of OOP

Inheritance

1. Inheritance is the process by which an object of one class acquires the properties of another class.
2. Reusable code
3. It resembles real life models.
4. Base class: The class which is inherited is called the base class.
5. Derived class: The class which inherits is called derived class.

Code

```
class Parent {
    public void print() {
        System.out.println("This is a function of Parent class.");
    }
}

class Child extends Parent {
    int x = 4;
}

public class Hello {

  public static void main(String args[]) {
    Child ch = new Child();

    ch.print();
  }
}
```

extends keyword is used to establish "Parent-Child" Relationship.

The print() is not defined in the Child class.
It is defined in the Parent class.

Encapsulation

1. Data and the methods which operate on that data are defined inside a single unit.
   This concept is called encapsulation.
2. No manipulation or access is allowed directly from outside the capsule or class.

Code

```
public class Person {

    // Private instance variable
    private String name;

    // Constructor
    public Person(String name) {
        this.name = name;
    }
    // Public getter for name
    public String getName() {
        return name;
    }

    // Public setter for name
    public void setName(String name) {
        this.name = name;
    }
    public static void main(String[] args) {
        Person person = new Person("Alice");
        System.out.println(person.getName()); // Alice

        person.setName("Bob");

        System.out.println(person.getName()); // Bob
    }
}
```

Any access/modification to the "name" attribute must happen through these two methods. No other way!!

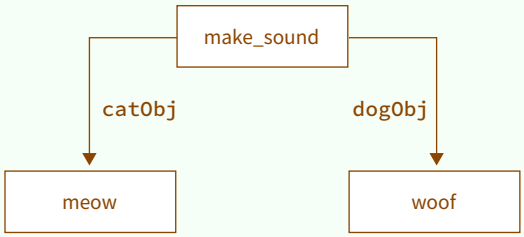
Polymorphism

1. Polymorphism is the capability of a method, class, or object to take on multiple forms.
2. It primarily manifests through:
   a. Inheritance
   b. Interfaces
   c. Method Overriding & Overloading

Code

```
interface Animal {
    String sound();

    public static void makeSound(Animal animal) {
        System.out.println(animal.sound());
    }
}

class Cat implements Animal {
    @Override
    public String sound() {
        return "meow";
    }
}

class Dog implements Animal {
    @Override
    public String sound() {
        return "woof";
    }
}

public class Main {

    public static void main(String[] args) {
        Animal catObj = new Cat();
        Animal dogObj = new Dog();

        Animal.makeSound(catObj);  // Output: meow
        Animal.makeSound(dogObj);  // Output: woof
    }
}
```

PolymorphimSound changes based on type of Animal passed.



The print() is not defined in the Child class.
It is defined in the Parent class.

Errors and Exception Handling

Syntax Errors

Code

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

```
error: '{' expected
public class HelloWorld

* Detected at Compile time.
```

Exceptions

Code

```
public class DivisionException {

  public static void main(String[] args) {
    int num = 10;
    int divisor = 0;
    int result = num / divisor;
  }
}
```

```
java.lang.ArithmeticException: / by zero

* Detected at run time. Cannot detect at Compile time.
```

Try...Catch

Code

```
int num = 10;
int divisor = 0;

try {
    int result = num / divisor;
    System.out.println("Result: " + result);
}
catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero!");
}
```

java.lang.ArithmeticException occurs and control moves to catch block.

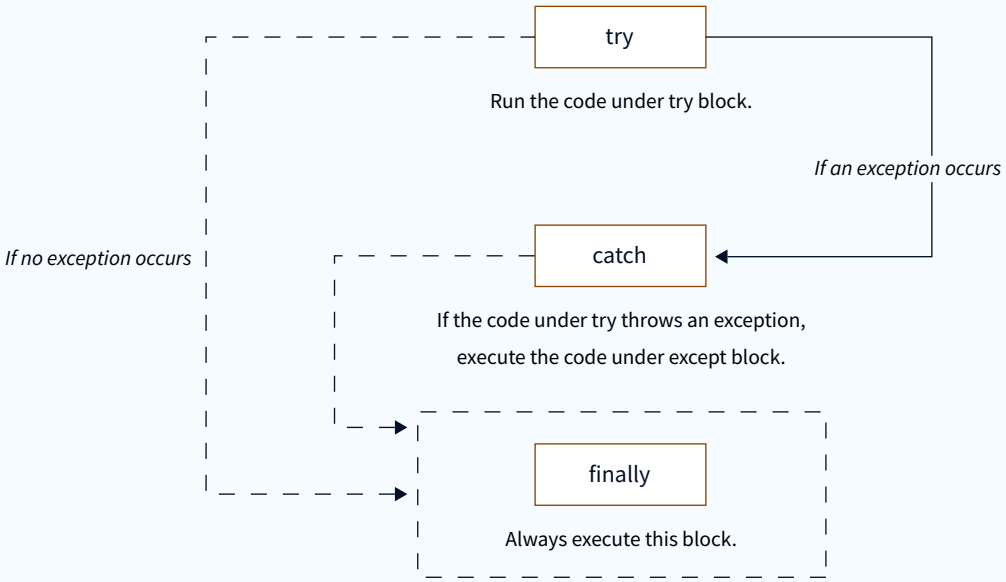
The Finally Clause

The code block under finally is always executed.

Code

```
try {
    int result = num / divisor;
    System.out.println("Result: " + result);
}
catch (ArithmeticException e) {
    System.out.println("Cannot divide by zero!");
}
finally {
    System.out.println("This block is always executed");
}
```

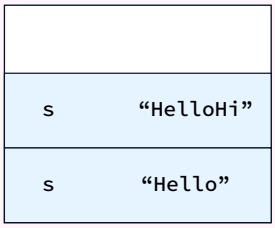
"finally" block is always executed!!



String, StringBuffer and StringBuilder

Immutable
-cannot be changed

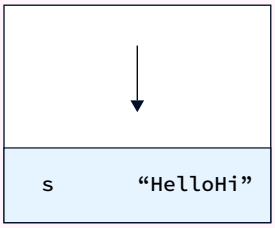
```
String s=new String
("Hello"); s+= "Hi";
```



String *Memory*

Mutable
-can be changed

```
StringBuffer s=new
StringBuffer("Hello");

s.append("Hi");
```



StringBuffer/
StringBuilder *Memory*

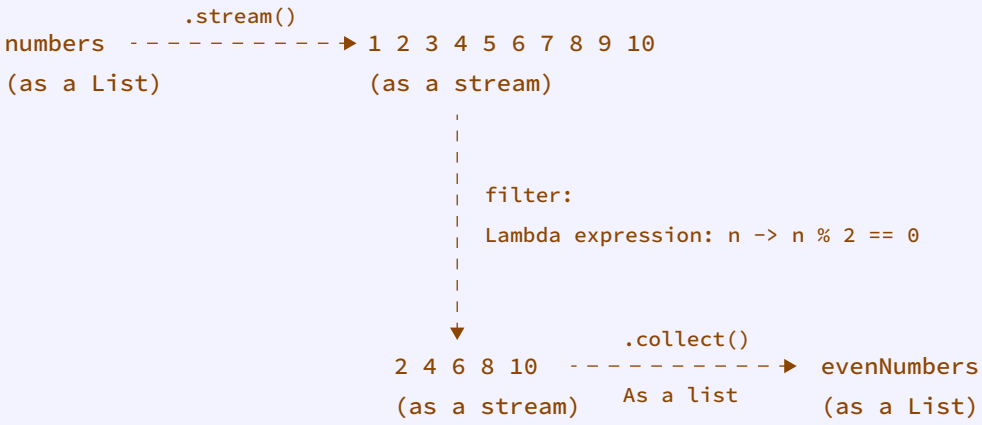
Index	String	String Buffer	String Builder
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No(Immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Thread Safe	Fast	Very slow	Fast

For more details, visit: https://www.scaler.com/topics/string-class-in-java/
https://www.scaler.com/topics/java/stringbuffer-in-java/
https://www.scaler.com/topics/java/stringbuilder-in-java/

Java Streams

1. A stream represents a sequence of elements and supports various operations on these elements.
2. Intermediate Operations: Transform a stream into another stream, e.g., filter, map, and sorted. They are always lazily executed.
3. Terminal Operations: When the intermediate operation is complete, the resultant stream is produced using methods like collect, reduce, toArray, etc.
4. One of the significant advantages of the Stream API is its inherent ability to parallelize operations.

```
List<Integer> numbers = Arrays.asList(1, 2, 3, 4, 5, 6, 7, 8, 9, 10);

List<Integer> evenNumbers = numbers.stream()
    .filter(n -> n % 2 == 0)  - - - - - - - - - - - - - - -> Lambda expression
    .collect(Collectors.toList());
```



Synchronized Method & Blocks

Synchronized Method

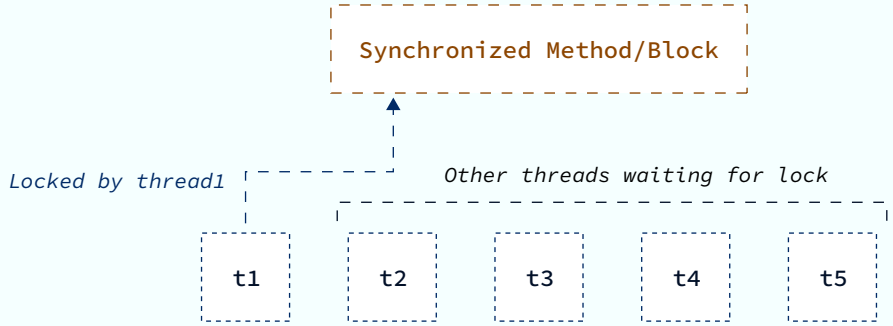
* A method that is declared as synchronized ensures that at most one thread can execute this method at any given time on the same object.
* It's achieved by putting a lock on the object for which the method is called.

```
public synchronized void synchronizedMethod() {
    // method body
}
```

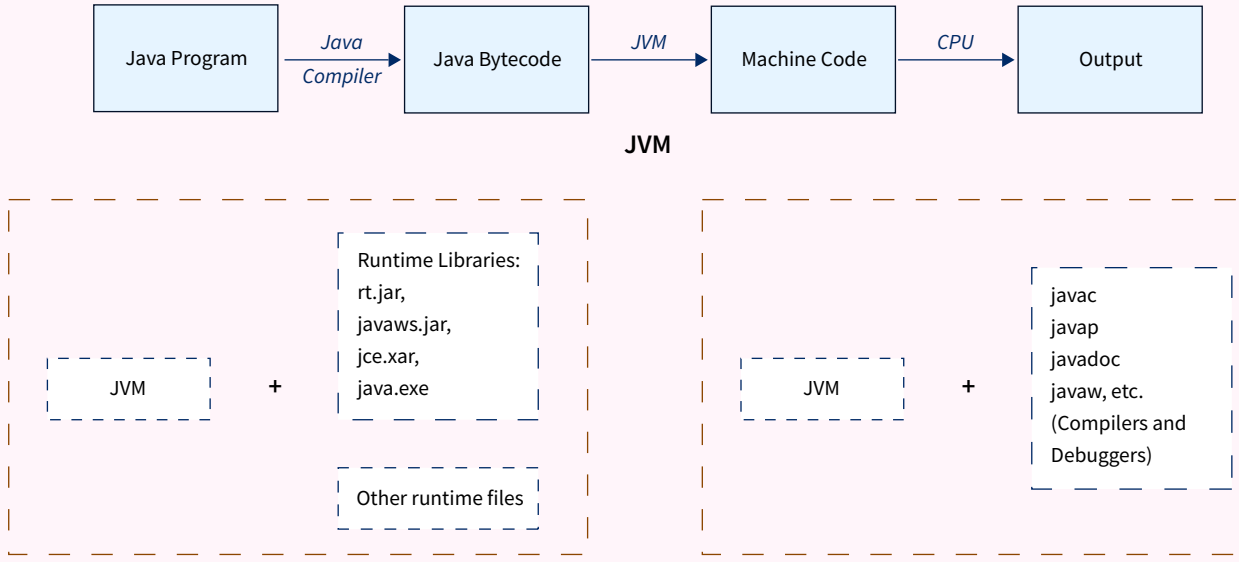
Synchronized Block

* Sometimes, synchronizing the entire method might be overkill. In such cases, you can use a synchronized block to only lock that specific section of the code.
* A synchronized block requires an object. The object's lock will be acquired by the block.

```
public void someMethod() {
    // non-critical section

    synchronized(lockObject) {
        // critical section
    }

    // other non-critical section
}
```



JVM, JRE and JDK



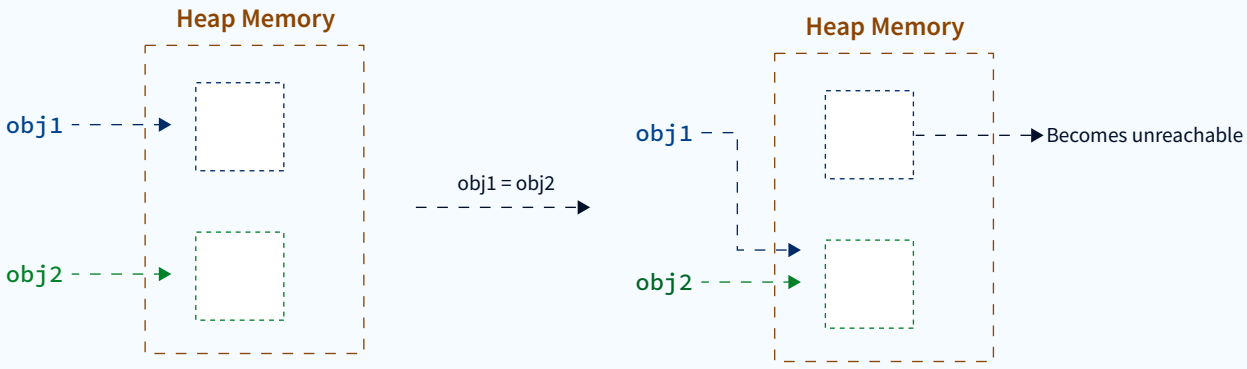
Garbage Collection

1. Automatic Memory Reclamation: Frees memory from unreferenced objects automatically.
2. Performance Overhead: Can introduce occasional application pauses.
3. Memory Efficiency: Garbage collection ensures that non-reachable allocated memory is freed.

Code

```
MyClass obj1 = new MyClass("Object 1");
MyClass obj2 = new MyClass("Object 2");

obj1 = obj2;
```

After this, original "obj1" object becomes unreachable.

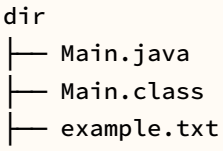


File Handling

Create a file

Code

```
try {
    File file = new File("example.txt");
    file.createNewFile();
} catch (IOException e) {
    e.printStackTrace();
}
```

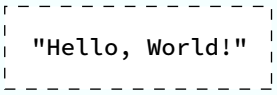


Writing to a file

Code

```
try {
    FileWriter writer = new FileWriter("example.txt");
    writer.write("Hello, World!");
    writer.close();
}
catch (IOException e) {
    e.printStackTrace();
}
```

"example.txt" file content



Reading from a file

Code

```
try {
    File file = new File("example.txt");
    BufferedReader br = new BufferedReader(new FileReader(file));
    String line;
    while ((line = br.readLine()) != null) {
        System.out.println(line);
    }
    br.close();
}
catch (IOException e) {
    e.printStackTrace();
}
```

File Content

Output

Hello, world!

SCALER TOPICS

Unlock your potential in software development with
FREE COURSES from **SCALER TOPICS!**

Register now and take the first step towards your future Success!

C++ for Beginners

5.9k enrolled Free

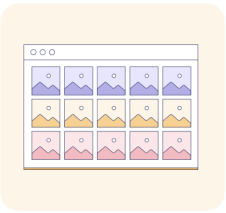
Java for Beginners

6.8k enrolled Free

That's not it. Explore 20+ Courses by clicking below

Explore Other Courses

Practice **CHALLENGES**
and become 1% better everyday



CIFAR-10 Image Classification Using PyTorch

Article

No. Of Questions : 3

Go to Challenge >

How to Build a Snake Game in JavaScript?

Article

No. Of Questions : 3

Go to Challenge >

Explore Other Challenges