Java

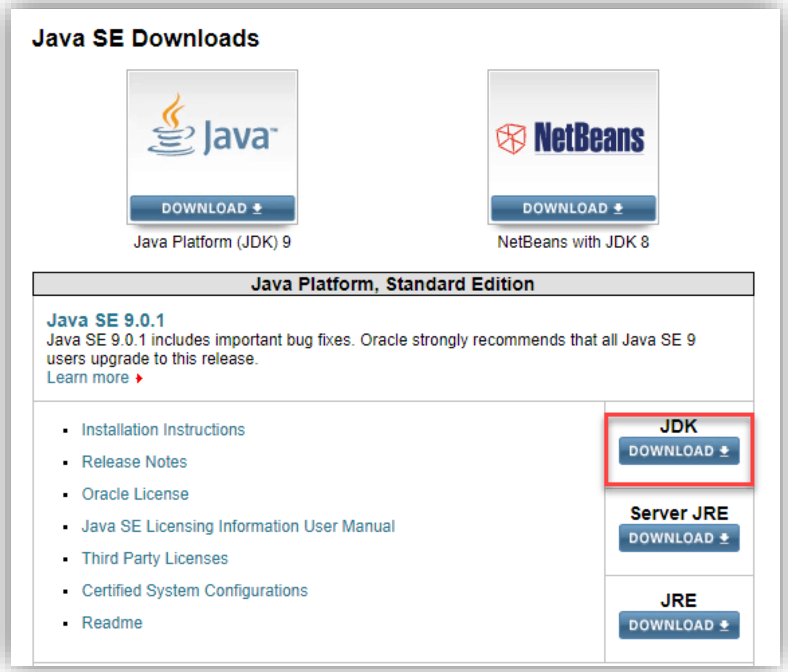
Agenda

- Download and install Java
- Download and install Eclipse IDE
- Java Variables
- Java Data Types
- Java Operators

Download and install Java

1) Download Link

- http://download.oracle.com/otn-pub/java/jdk/9.0.4+11/c2514751926b4512b076cc82f959763f/jdk-9.0.4_windows-x64_bin.exe



Java SE Downloads

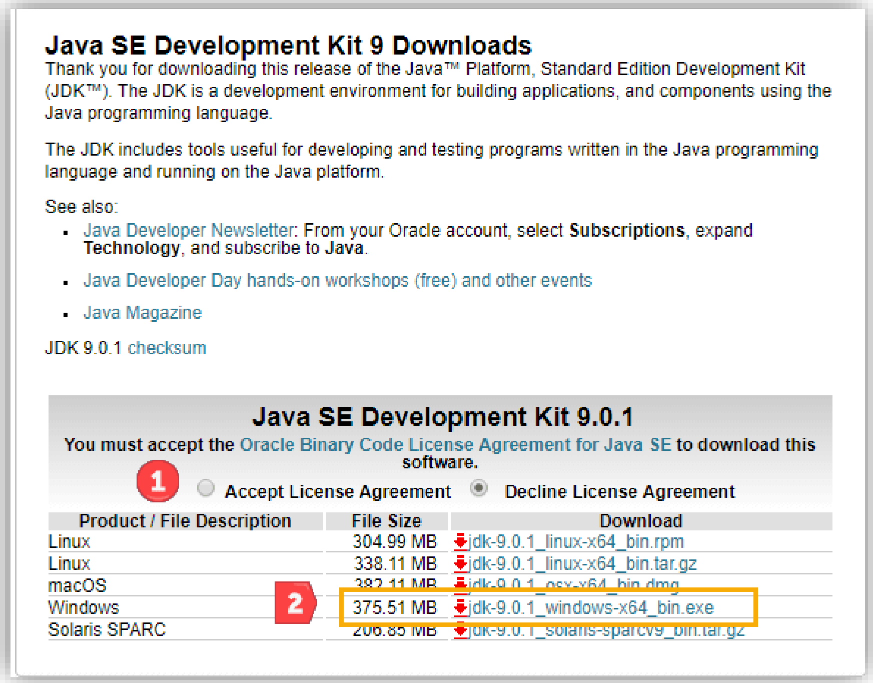
Java Platform (JDK) 9

NetBeans with JDK 8

Java Platform, Standard Edition

Java SE 9.0.1
Java SE 9.0.1 includes important bug fixes. Oracle strongly recommends that all Java SE 9 users upgrade to this release.
Learn more

- Installation Instructions
- Release Notes
- Oracle License
- Java SE Licensing Information User Manual
- Third Party Licenses
- Certified System Configurations
- Readme



Java SE Development Kit 9 Downloads

Thank you for downloading this release of the Java™ Platform, Standard Edition Development Kit (JDK™). The JDK is a development environment for building applications, and components using the Java programming language.

The JDK includes tools useful for developing and testing programs written in the Java programming language and running on the Java platform.

See also:

- Java Developer Newsletter: From your Oracle account, select **Subscriptions**, expand **Technology**, and subscribe to **Java**.
- Java Developer Day hands-on workshops (free) and other events
- Java Magazine

JDK 9.0.1 checksum

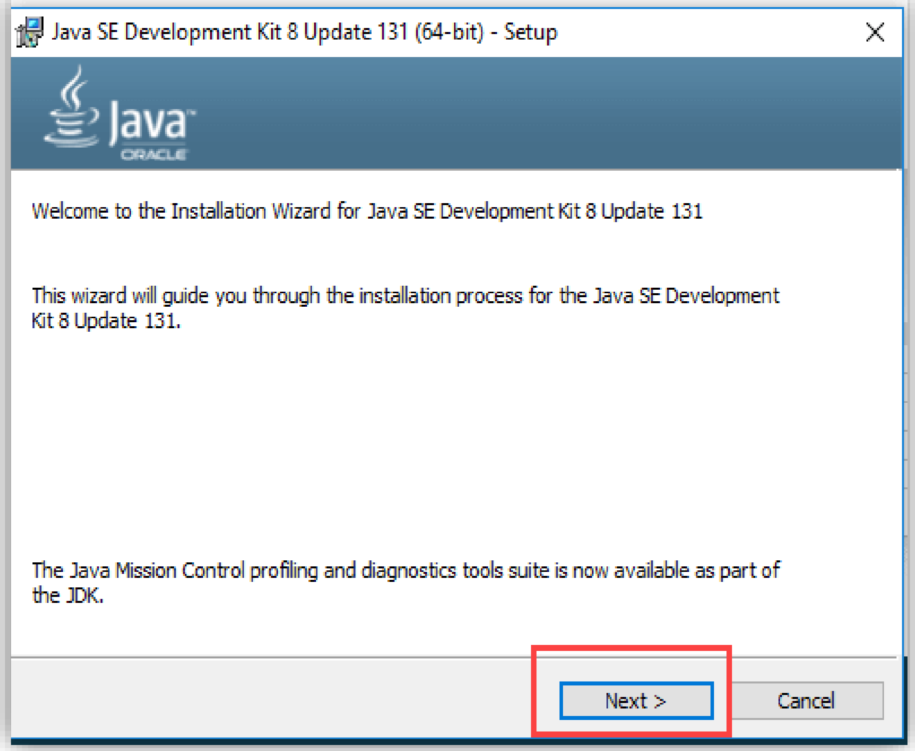
Java SE Development Kit 9.0.1

You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

○ Accept License Agreement ● Decline License Agreement

Product / File Description	File Size	Download
Linux	304.99 MB	jdk-9.0.1_linux-x64_bin.rpm
Linux	338.11 MB	jdk-9.0.1_linux-x64_bin.tar.gz
macOS	382.11 MB	jdk-9.0.1_osx-x64_bin.dmg
Windows	375.51 MB	jdk-9.0.1_windows-x64_bin.exe
Solaris SPARC	206.85 MB	jdk-9.0.1_solaris-sparcv9_bin.tar.gz

2) Once the download is complete, run the exe for install JDK. Click Next

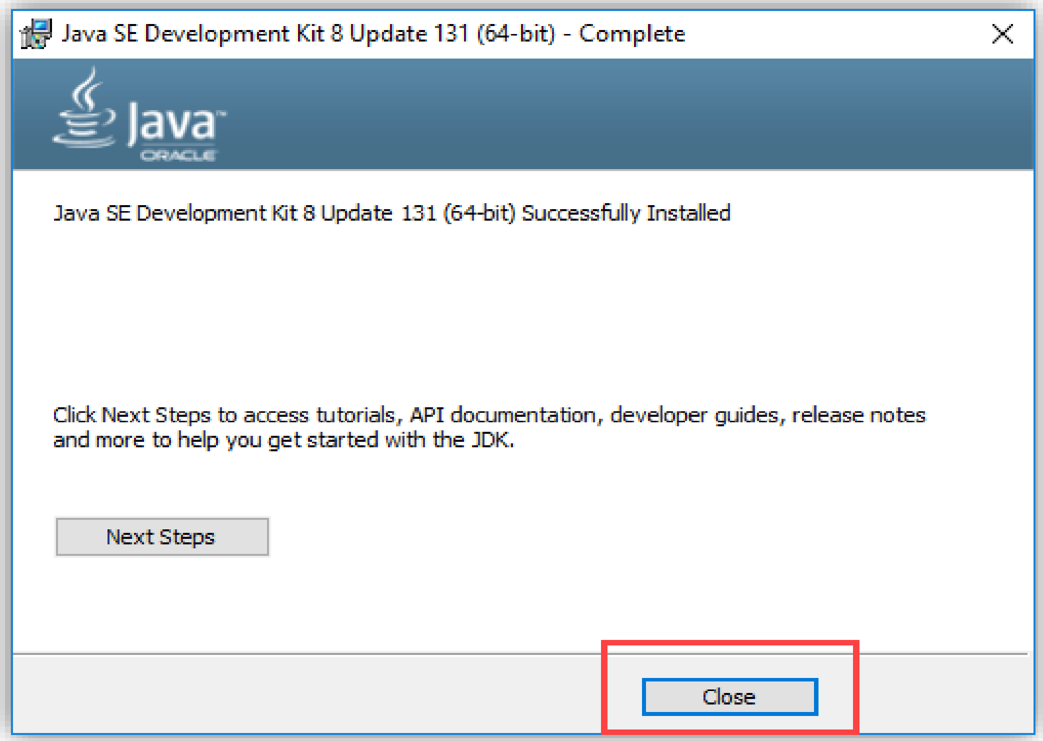


Java SE Development Kit 8 Update 131 (64-bit) - Setup

Welcome to the Installation Wizard for Java SE Development Kit 8 Update 131

This wizard will guide you through the installation process for the Java SE Development Kit 8 Update 131.

The Java Mission Control profiling and diagnostics tools suite is now available as part of the JDK.

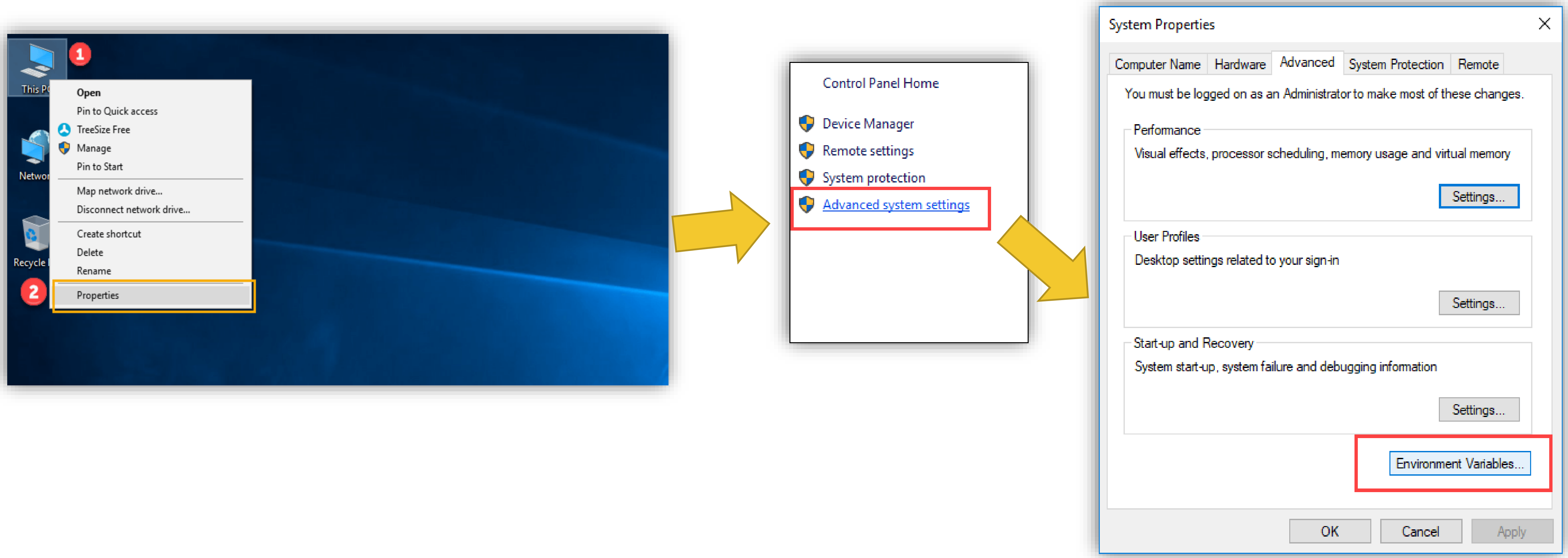


Java SE Development Kit 8 Update 131 (64-bit) - Complete

Java SE Development Kit 8 Update 131 (64-bit) Successfully Installed

Click Next Steps to access tutorials, API documentation, developer guides, release notes and more to help you get started with the JDK.

Set Java Environment in Java



You must be logged on as an Administrator to make most of these changes.

Performance: Visual effects, processor scheduling, memory usage and virtual memory

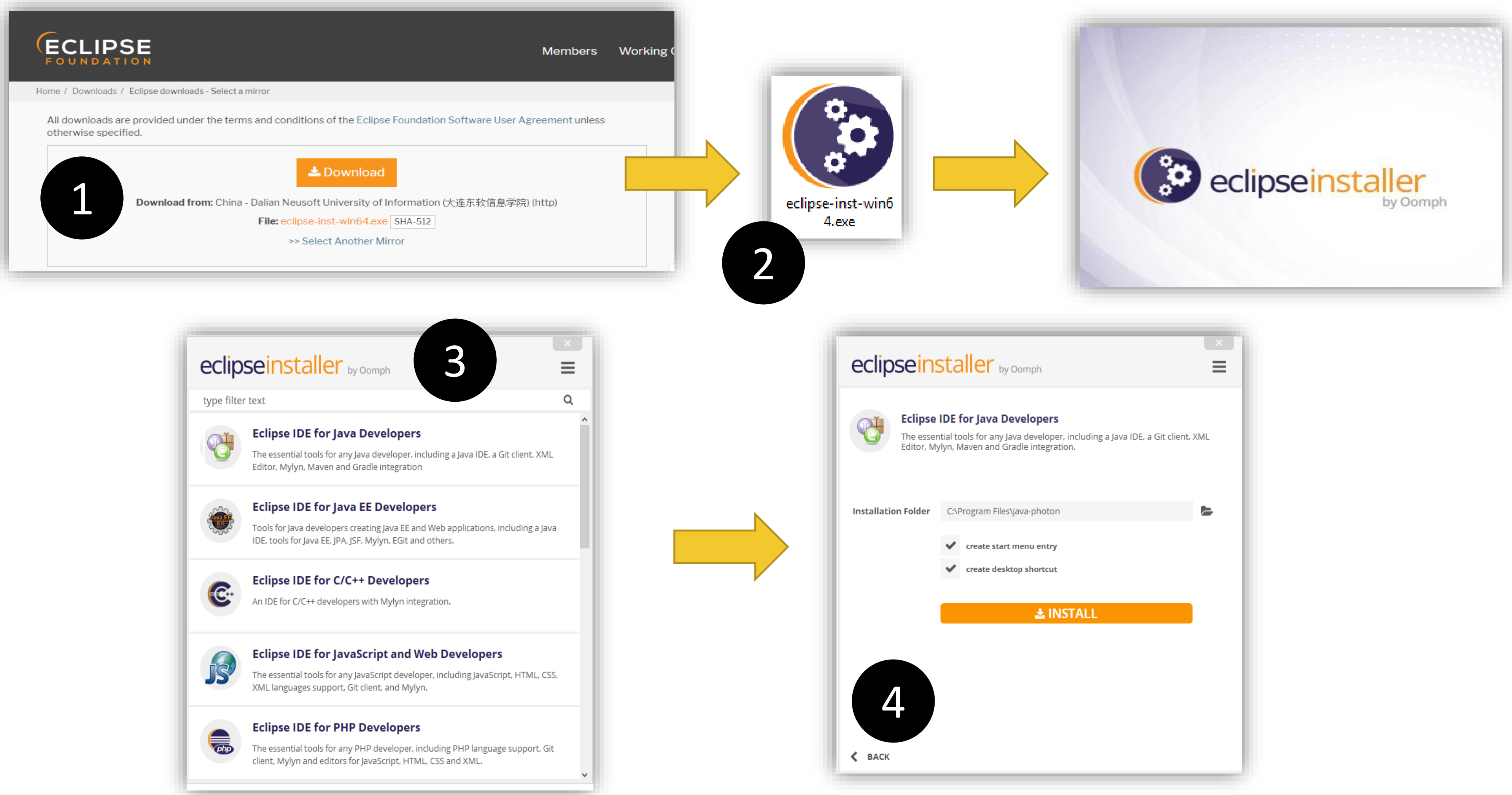
User Profiles: Desktop settings related to your sign-in

Start-up and Recovery: System start-up, system failure and debugging information

Eclipse IDE

- https://www.eclipse.org/downloads/download.php?file=/oomph/epp/photon/R/eclipse-inst-win64.exe

Download and Start Eclipse IDE



1

ECLIPSE FOUNDATION

Members Working

Home / Downloads / Eclipse downloads - Select a mirror

All downloads are provided under the terms and conditions of the Eclipse Foundation Software User Agreement unless otherwise specified.

Download

Download from: China - Dalian Neusoft University of Information (大连东软信息学院) (http)

File: eclipse-inst-win64.exe SHA-512

>> Select Another Mirror

2

eclipse-inst-win64.exe

eclipseinstaller by Oomph

3

eclipseinstaller by Oomph

type filter text

Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration

Eclipse IDE for Java EE Developers
Tools for Java developers creating Java EE and Web applications, including a Java IDE, tools for Java EE, JPA, JSF, Mylyn, EGit and others.

Eclipse IDE for C/C++ Developers
An IDE for C/C++ developers with Mylyn integration.

Eclipse IDE for JavaScript and Web Developers
The essential tools for any JavaScript developer, including JavaScript, HTML, CSS, XML languages support, Git client, and Mylyn.

Eclipse IDE for PHP Developers
The essential tools for any PHP developer, including PHP language support, Git client, Mylyn and editors for JavaScript, HTML, CSS and XML.

4

eclipseinstaller by Oomph

Eclipse IDE for Java Developers
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

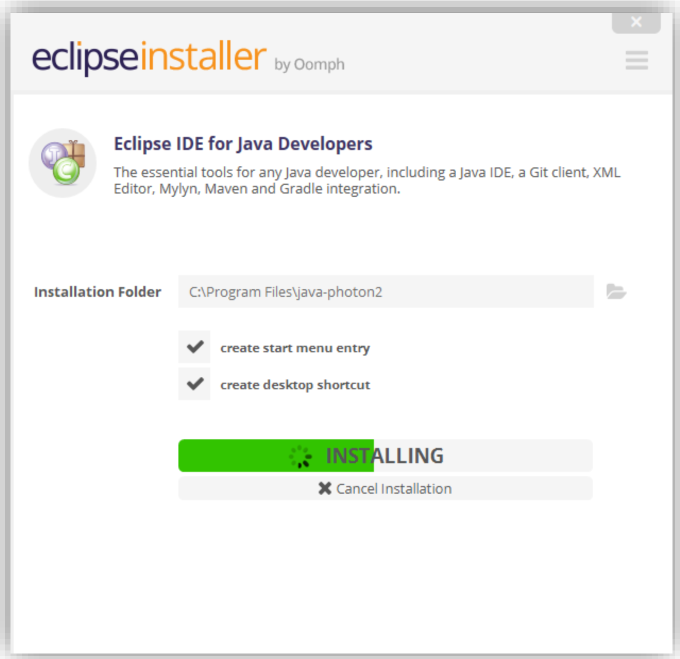
Installation Folder C:\Program Files\java-photon

✓ create start menu entry

✓ create desktop shortcut

INSTALL

BACK



eclipseinstaller by Oomph

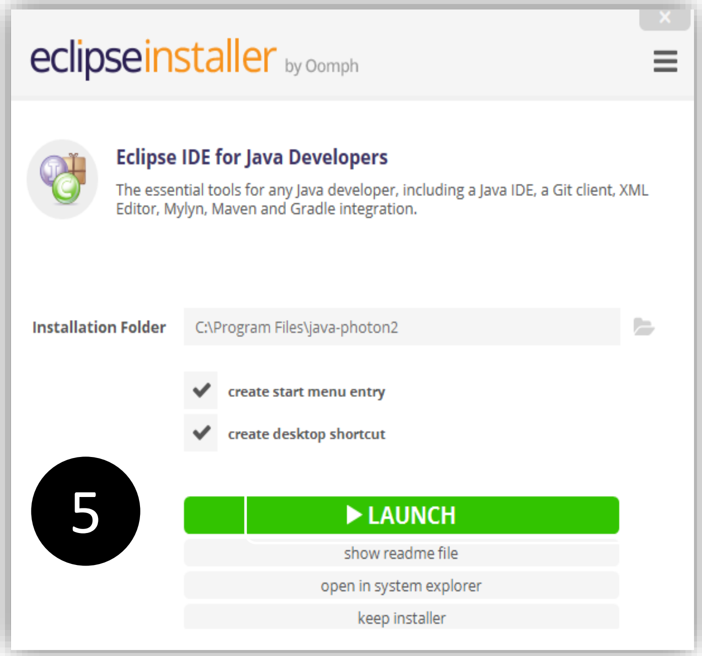
Eclipse IDE for Java Developers

The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Installation Folder: C:\Program Files\java-photon2

☑ create start menu entry

☑ create desktop shortcut



eclipseinstaller by Oomph

Eclipse IDE for Java Developers

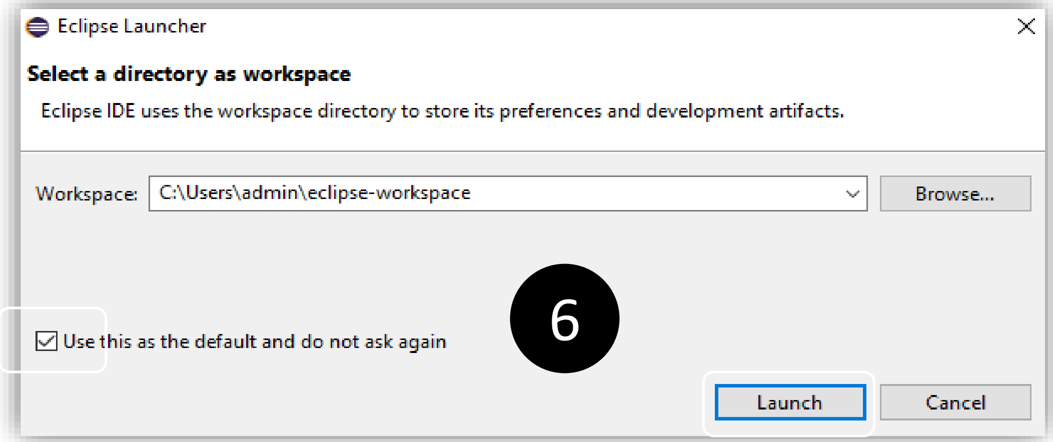
The essential tools for any Java developer, including a Java IDE, a Git client, XML Editor, Mylyn, Maven and Gradle integration.

Installation Folder: C:\Program Files\java-photon2

☑ create start menu entry

☑ create desktop shortcut

5



Eclipse Launcher

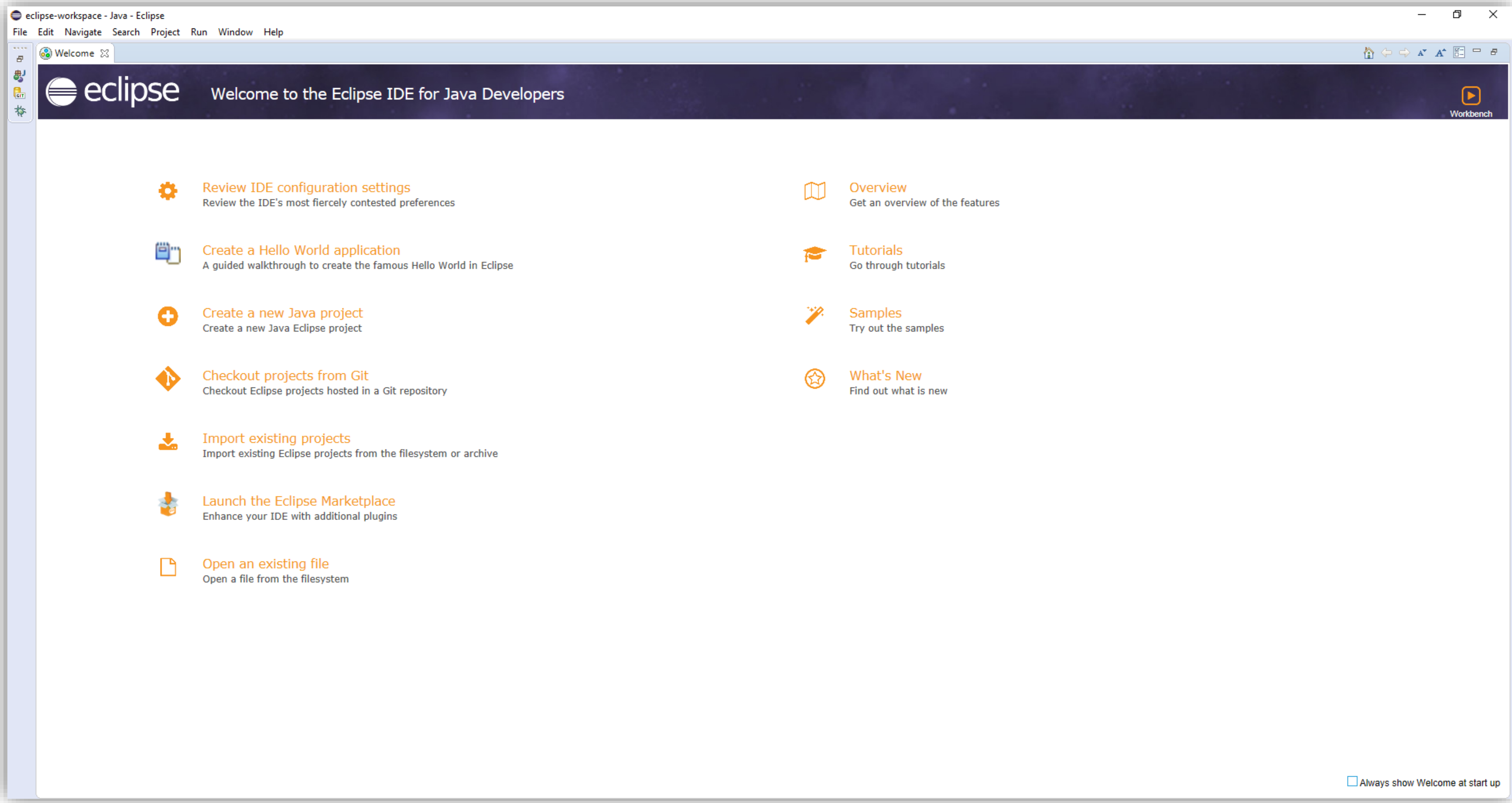
Select a directory as workspace

Eclipse IDE uses the workspace directory to store its preferences and development artifacts.

Workspace: C:\Users\admin\eclipse-workspace

☑ Use this as the default and do not ask again

6



eclipse-workspace - Java - Eclipse
File Edit Navigate Search Project Run Window Help
Welcome
eclipse
Welcome to the Eclipse IDE for Java Developers
Workbench

Review IDE configuration settings
Review the IDE's most fiercely contested preferences

Create a Hello World application
A guided walkthrough to create the famous Hello World in Eclipse

Create a new Java project
Create a new Java Eclipse project

Checkout projects from Git
Checkout Eclipse projects hosted in a Git repository

Import existing projects
Import existing Eclipse projects from the filesystem or archive

Launch the Eclipse Marketplace
Enhance your IDE with additional plugins

Open an existing file
Open a file from the filesystem

Overview
Get an overview of the features

Tutorials
Go through tutorials

Samples
Try out the samples

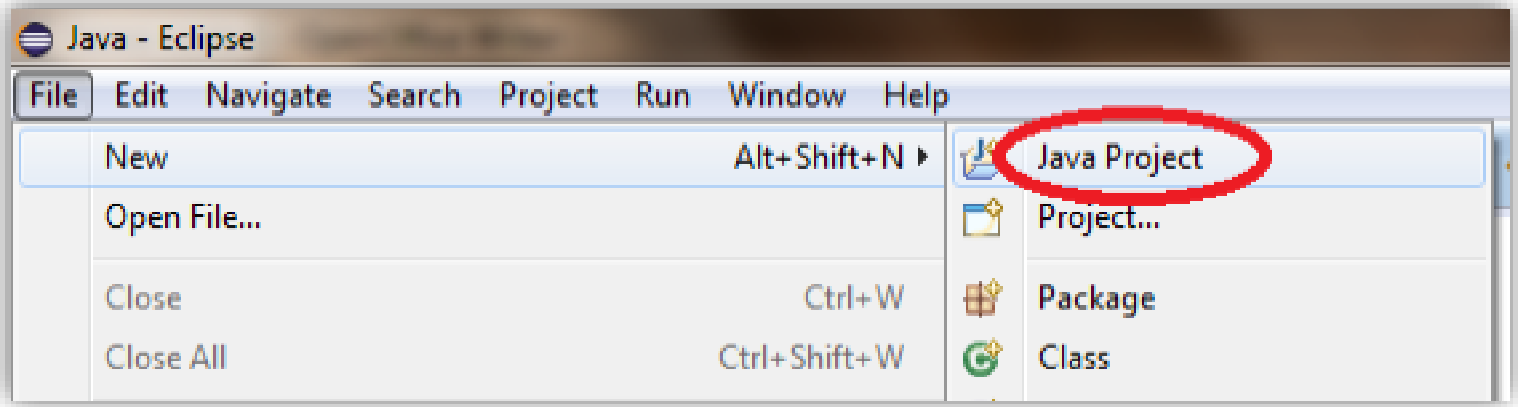
What's New
Find out what is new

□ Always show Welcome at start up

Create a Java Project

- To create a new Java project in Eclipse

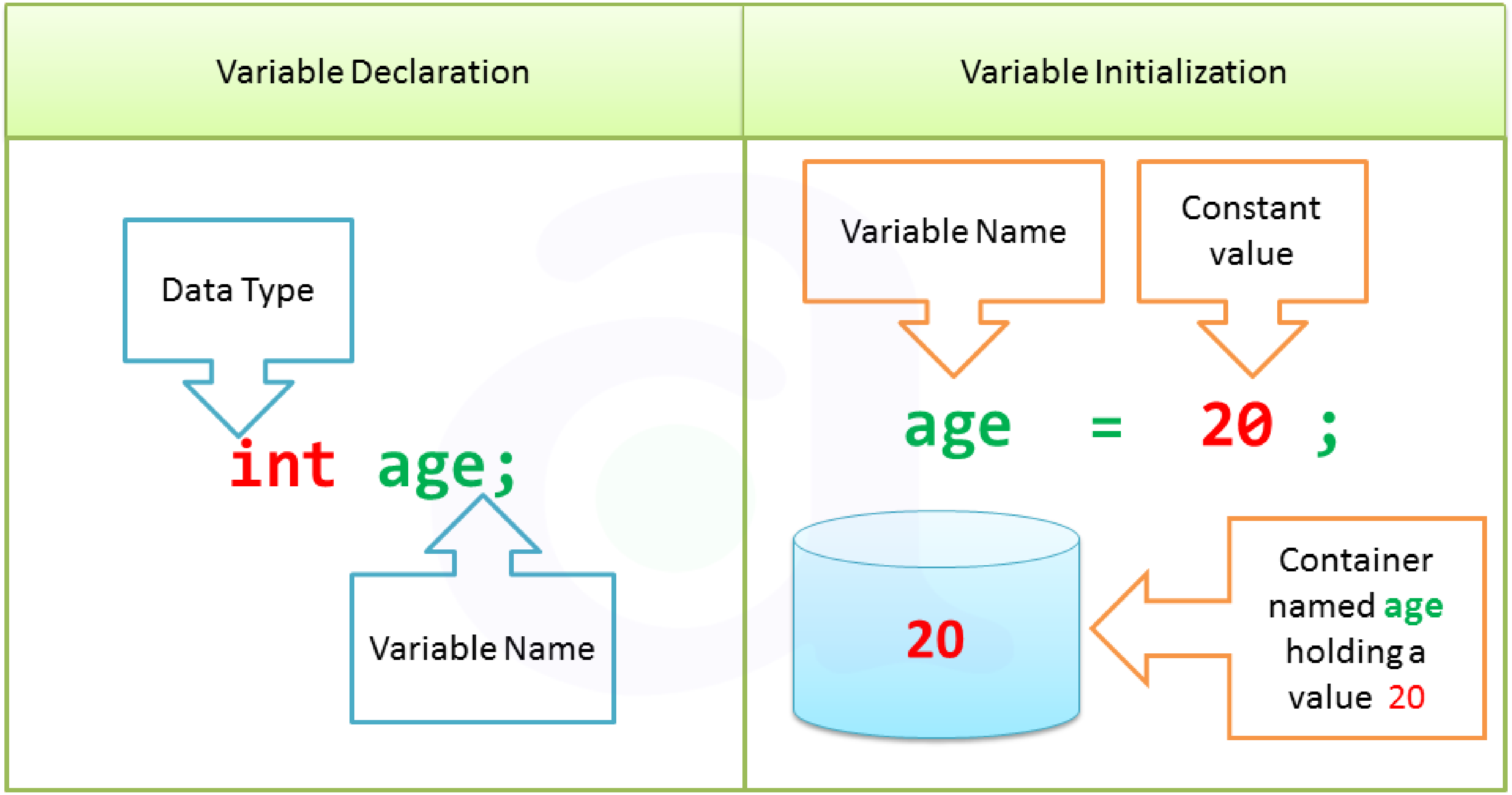
Go to **File → New → Java Project → Specify Project Name → Finish**



Java Variables

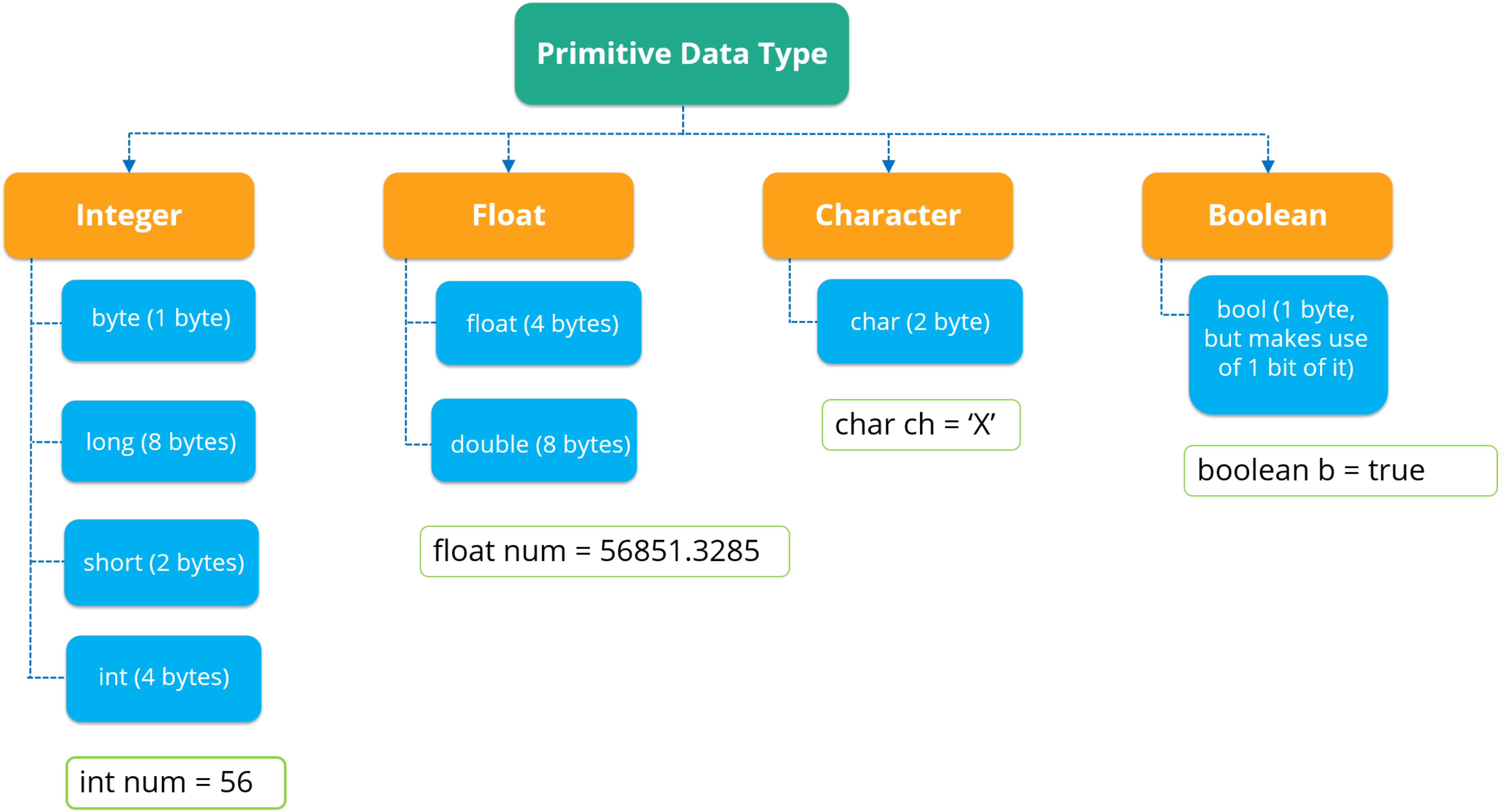
- **What is a Variable?**
- A variable is a container which holds value.
- Every variable is assigned a **data type** which designates the type and quantity of value it can hold.
- In order to use a variable in a program you to need to perform 2 steps
  - Variable Declaration
  - Variable Initialization

Java Variables(Cont..)



int variable Declaration and Initialization

Java Data Types



Primitive Data Type

Integer

byte (1 byte)

long (8 bytes)

short (2 bytes)

int (4 bytes)

int num = 56

Float

float (4 bytes)

double (8 bytes)

float num = 56851.3285

Character

char (2 byte)

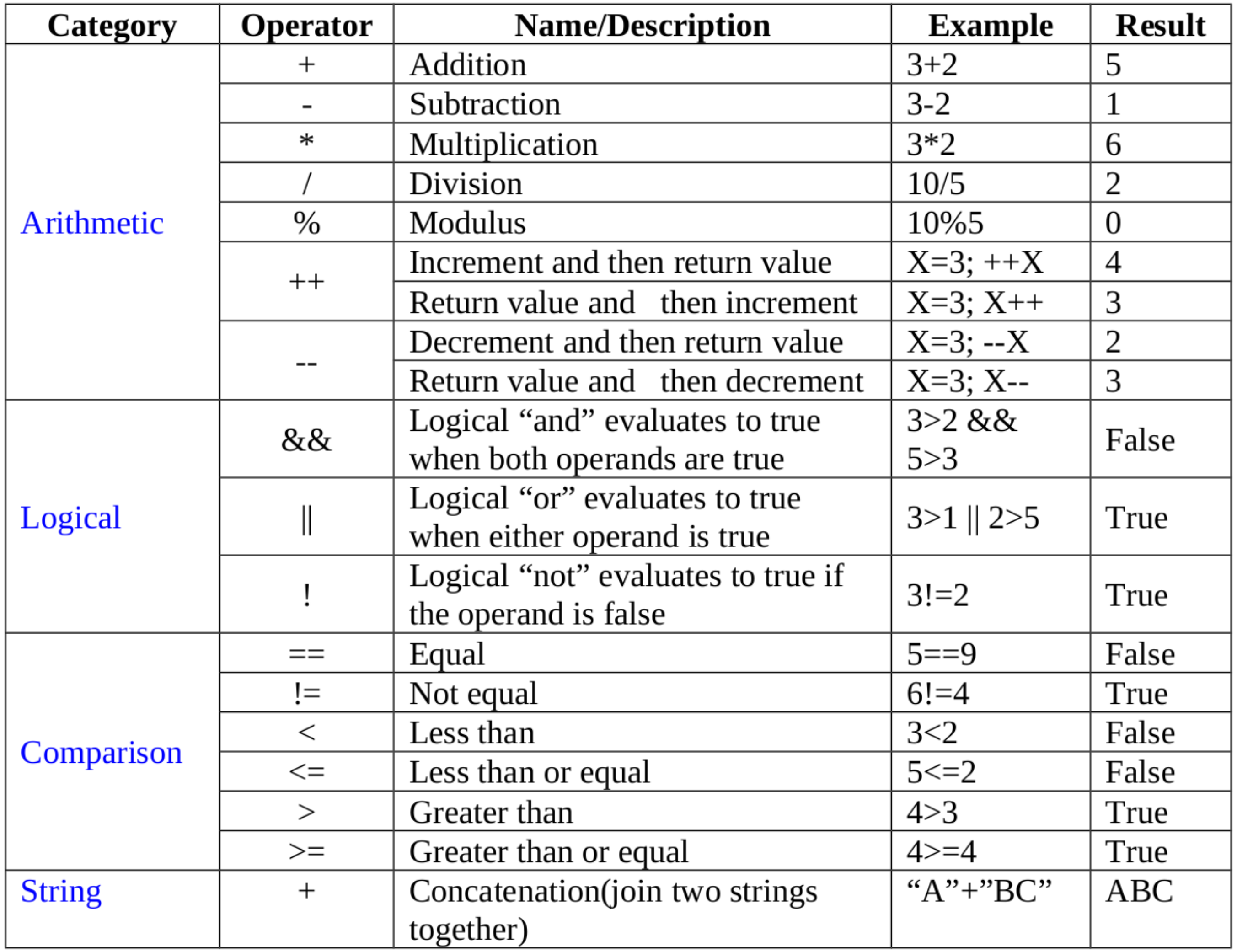
char ch = ‘X’

Boolean

bool (1 byte, but makes use of 1 bit of it)

boolean b = true

Java Operators



Category	Operator	Name/Description	Example	Result
Arithmetic	+	Addition	3+2	5
	-	Subtraction	3-2	1
	*	Multiplication	3*2	6
	/	Division	10/5	2
	%	Modulus	10%5	0
	++	Increment and then return value	X=3; ++X	4
		Return value and then increment	X=3; X++	3
	--	Decrement and then return value	X=3; --X	2
		Return value and then decrement	X=3; X--	3
Logical	&&	Logical “and” evaluates to true when both operands are true	3>2 && 5>3	False
	||	Logical “or” evaluates to true when either operand is true	3>1 || 2>5	True
	!	Logical “not” evaluates to true if the operand is false	3!=2	True
Comparison	==	Equal	5==9	False
	!=	Not equal	6!=4	True
	<	Less than	3<2	False
	<=	Less than or equal	5<=2	False
	>	Greater than	4>3	True
	>=	Greater than or equal	4>=4	True
String	+	Concatenation(join two strings together)	“A”+”BC”	ABC

Assignment

1. Write a Java program to print 'Hello' on screen and then print your name on a separate line.
   - Expected Output :
   - Hello
   - Pavan
2. Write a Java program to print the sum of two numbers.
   - Test Data: 4 + 36
   - Expected Output : 50

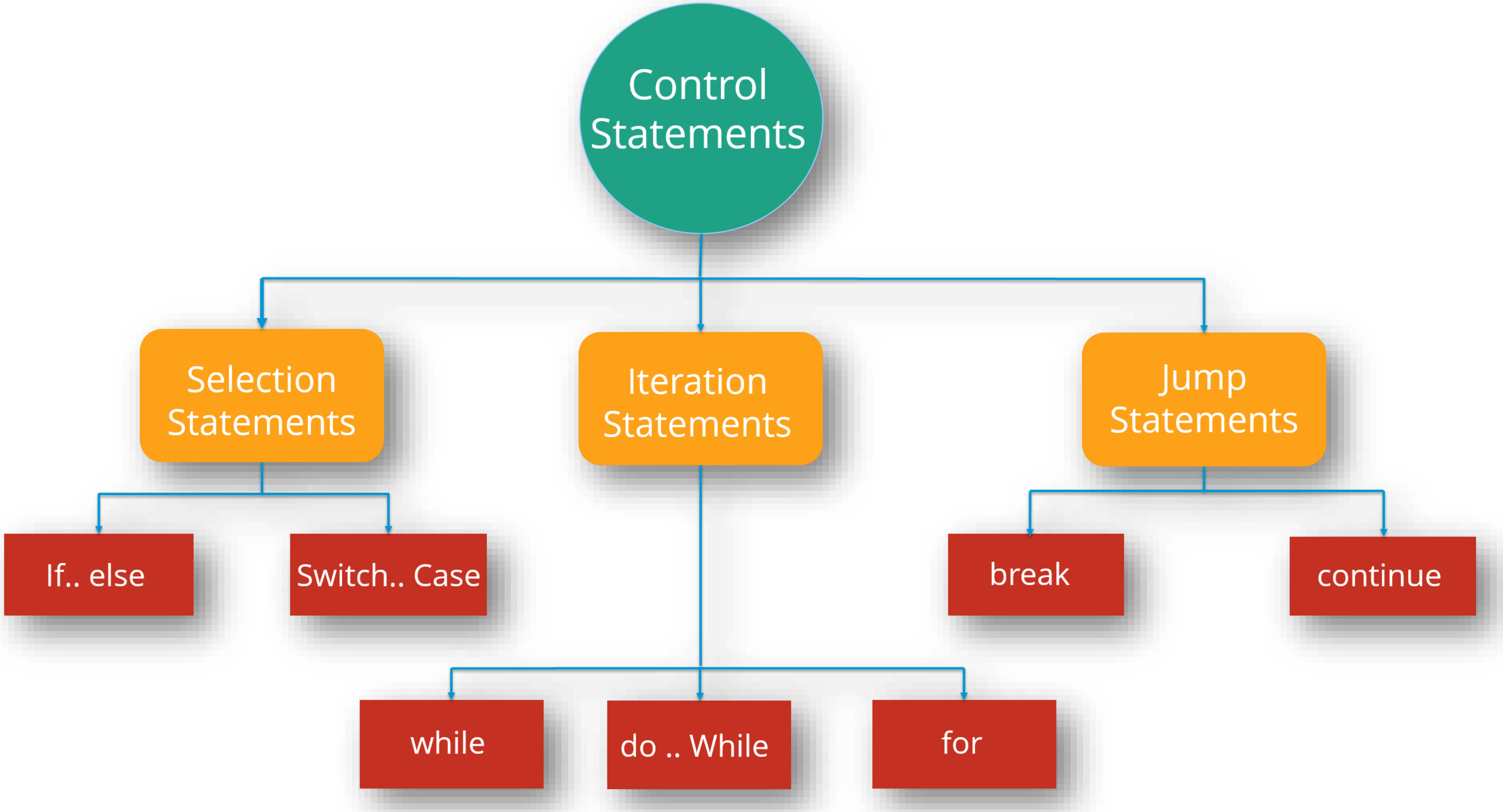
3. Write a Java program to swap two numbers.

4. Write a Java Program without using third variable.

Agenda

- Control Statements
- Conditional/Statements(Selection Statements)
- Loops /Iterative Statements
- Jump Statements

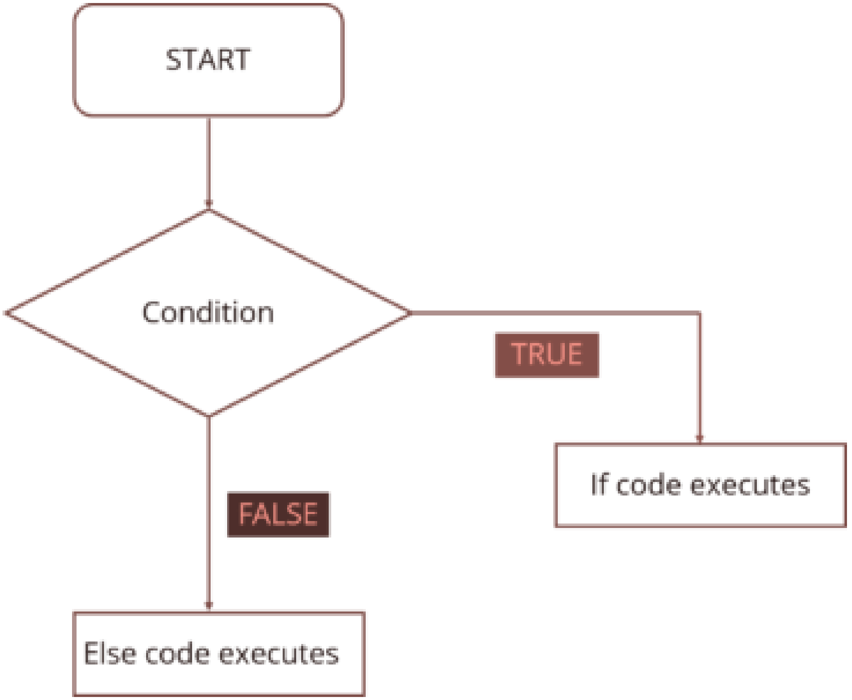
Control Statements

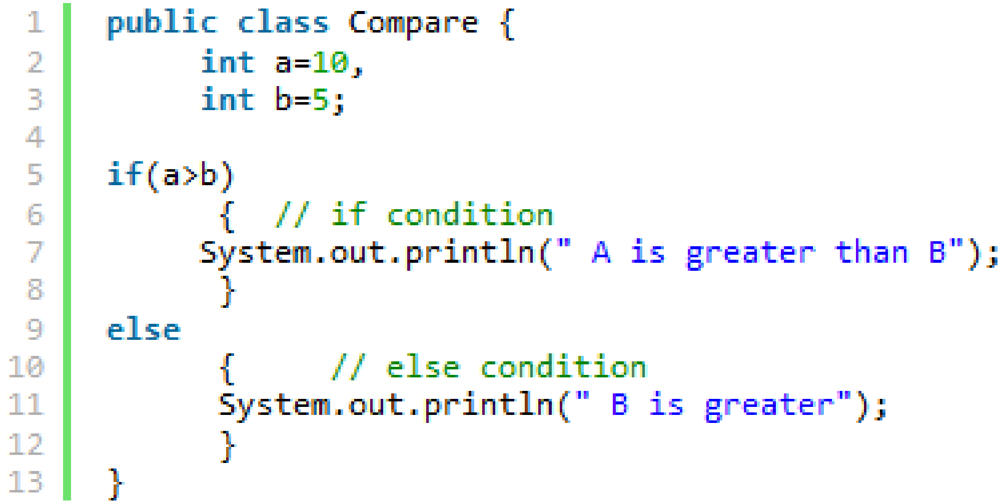


If..else

- In this flowchart, the code will respond in the following way:

1. First of all, it will enter the loop where it checks the condition.
2. If the condition is true, the set of statements in ‘if’ part will be executed.
3. If the condition is false, the set of statements in the ‘else’ part will be executed.





```java
public class Compare {
     int a=10,
     int b=5;

if(a>b)
      {  // if condition
     System.out.println(" A is greater than B");
      }
else
      {     // else condition
      System.out.println(" B is greater");
      }
}
```

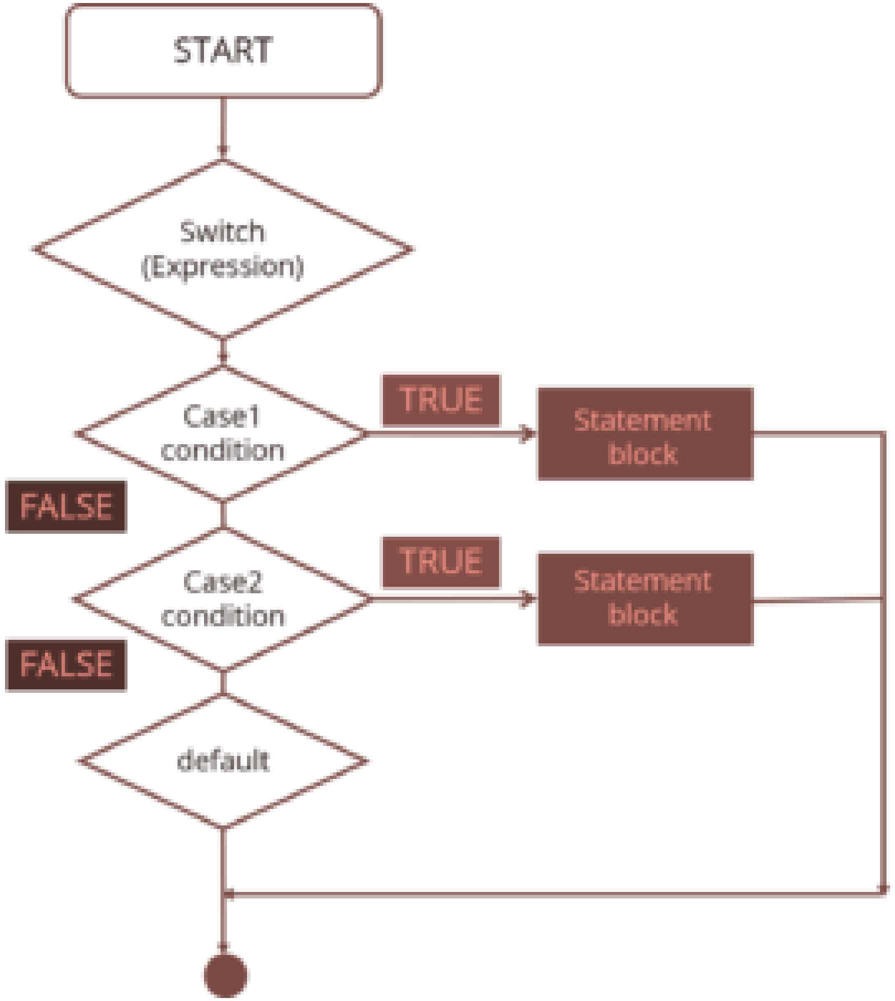
Switch..case

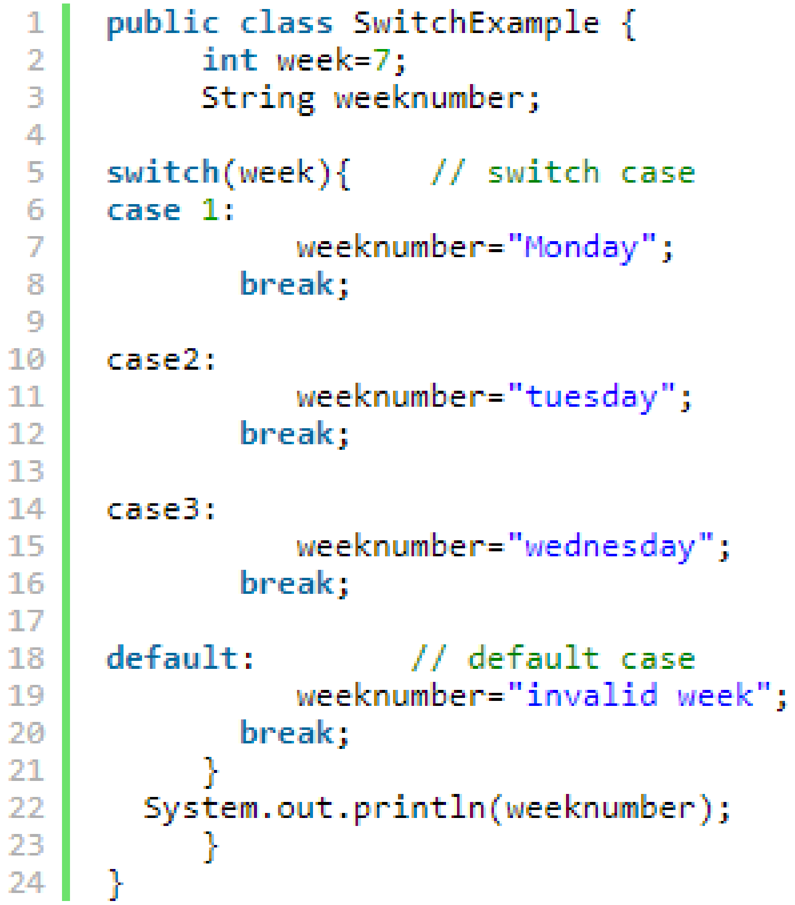
- The switch statement defines multiple paths for execution of a set of statements. It is a better alternative than using a large set of if-else statements as it is a multi-way branch statement.

- In this Switch case flowchart, the code will respond in the following steps:

  1. First of all it will enter the switch case which has an expression.
  2. Next it will go to Case 1 condition, checks the value passed to the condition. If it is true, Statement block will execute. After that, it will break from that switch case.
  3. In case it is false, then it will switch to the next case. If Case 2 condition is true, it will execute the statement and break from that case, else it will again jump to the next case.
  4. Now let’s say you have not specified any case or there is some wrong input from the user, then it will go to the default case where it will print your default statement.

Switch..case (Cont..)



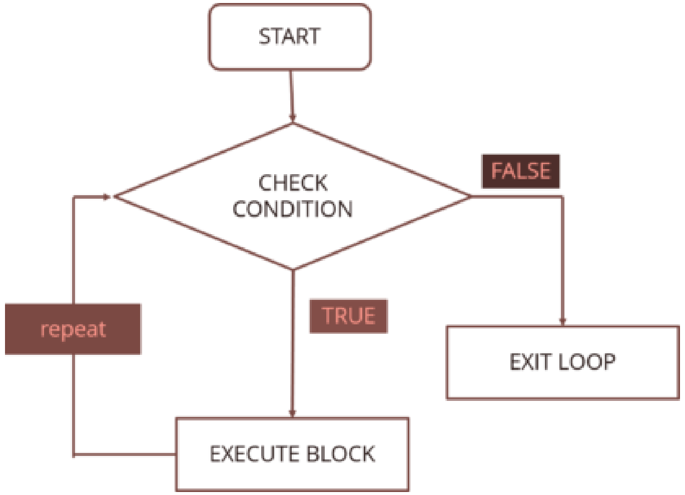


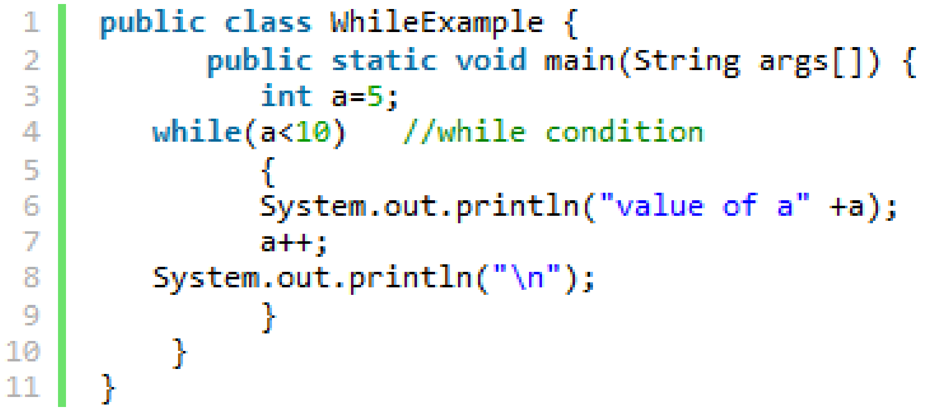
```
public class SwitchExample {
     int week=7;
     String weeknumber;

switch(week){     // switch case
case 1:
          weeknumber="Monday";
       break;

case2:
          weeknumber="tuesday";
       break;

case3:
          weeknumber="wednesday";
       break;

default:         // default case
          weeknumber="invalid week";
       break;
     }
  System.out.println(weeknumber);
     }
}
```

While loop

- **While statement:** Repeat a group of statements while a given condition is true. It tests the condition before executing the loop body.

- In this flowchart, the code will respond in the following steps:
  1. First of all, it will enter the loop where it checks the condition.
  2. If it’s true, it will execute the set of code and repeat the process.
  3. If it’s False, it will directly exit the loop.





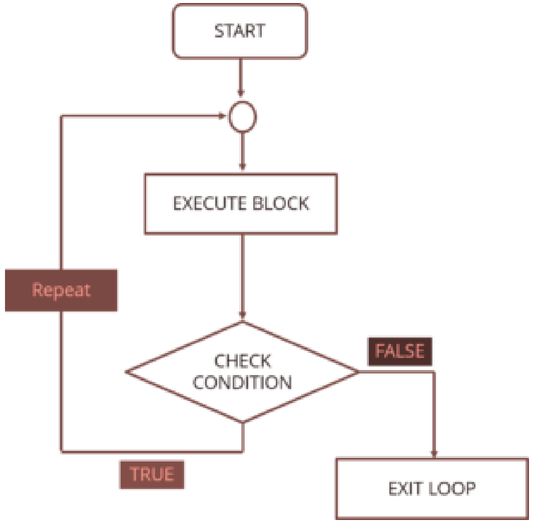
```
public class WhileExample {
        public static void main(String args[]) {
            int a=5;
    while(a<10)    //while condition
            {
            System.out.println("value of a" +a);
            a++;
    System.out.println("\n");
            }
      }
}
```

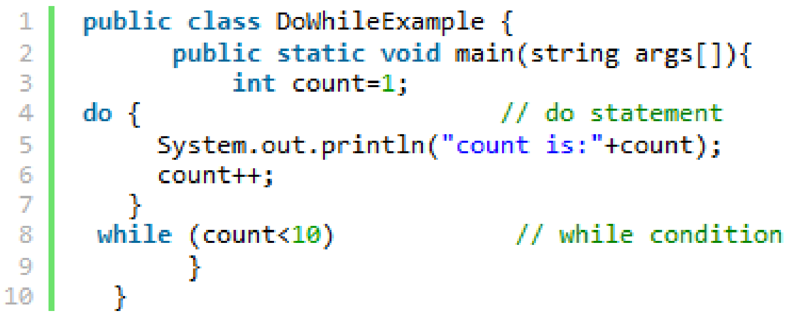
Do..while loop

- **Do-while statement:** It is like a while statement, but it tests the condition at the end of the loop body. Also, it will executes the program at least once.

- In this do-while flowchart, the code will respond in the following steps:

  1. First of all, it will execute a set of statements that is mentioned in your ‘do’ block.
  2. After that, it will come to ‘while’ part where it checks the condition.
  3. If the condition is true, it will go back and execute the statements.
  4. If the condition is false, it will directly exit the loop.



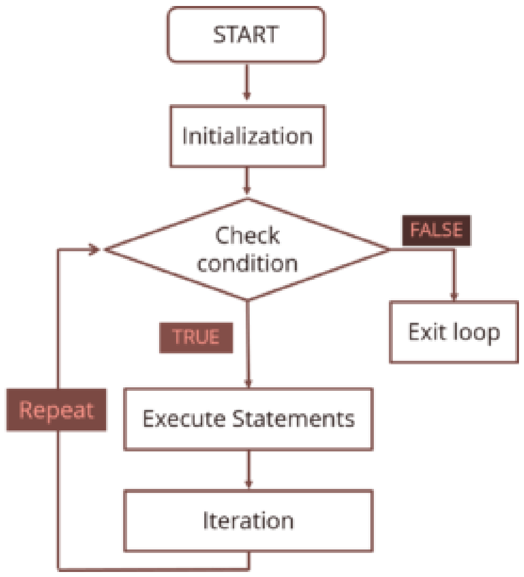


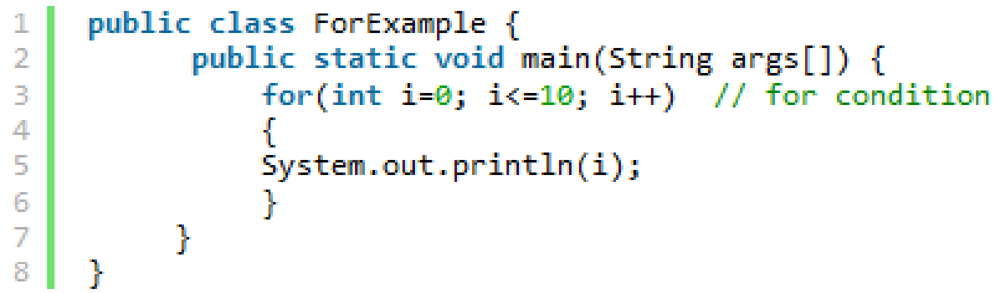
```
public class DoWhileExample {
      public static void main(string args[]){
          int count=1;
do {                            // do statement
     System.out.println("count is:"+count);
     count++;
   }
 while (count<10)              // while condition
       }
  }
```

For loop

- **For statement:** For statement execute a sequence of statements multiple time where you can manage the loop variable. You basically have 3 operations here: initialization, condition and iteration.

- In this flowchart, the code will respond in the following steps:
  1. First of all, it will enter the loop where it checks the condition.
  2. Next, if the condition is true, the statements will be executed.
  3. If the condition is false, it directly exits the loop.





```
public class ForExample {
        public static void main(String args[]) {
             for(int i=0; i<=10; i++)  // for condition
             {
             System.out.println(i);
             }
        }
}
```

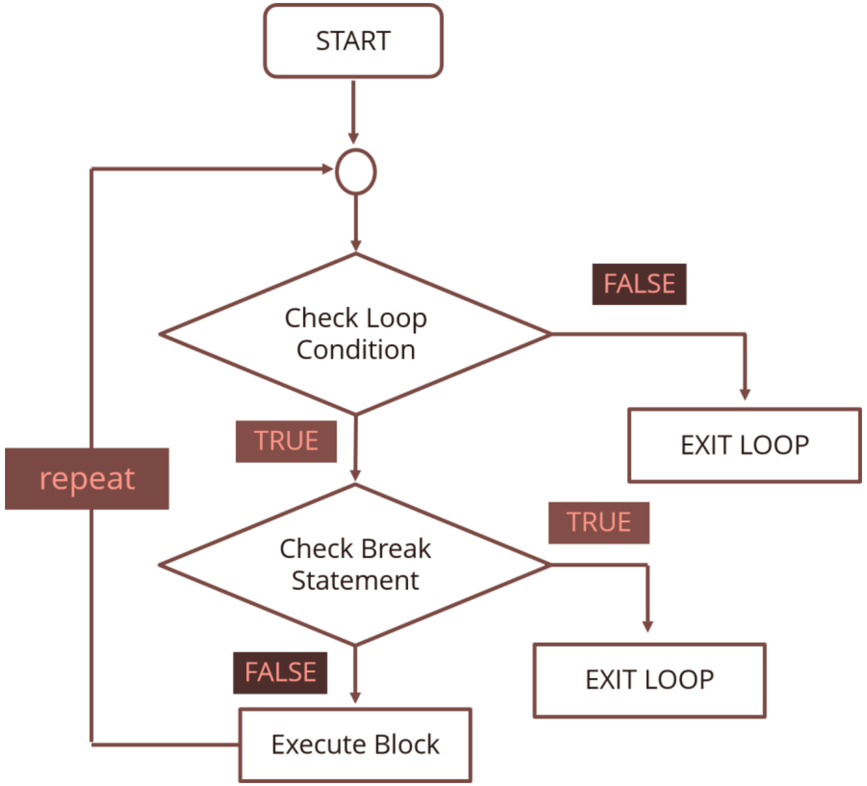
Jump Statements

- **Jump statement:** Jump statement are used to transfer the control to another part of your program. These are further classified into – *break* and *continue*.

Break statement

- **Break statement:** Whenever a break statement is used, the loop is terminated and the program control is resumed to the next statement following the loop

- In this flowchart, the code will respond in the following steps:
  1. First of all, it will enter the loop where it checks the condition.
  2. If the loop condition is false, it directly exits the loop.
  3. If the condition is true, it will then check the break condition.
  4. If break condition is true, it exists from the loop.
  5. If the break condition is false, then it will execute the statements that are remaining in the loop and then repeat the same steps.

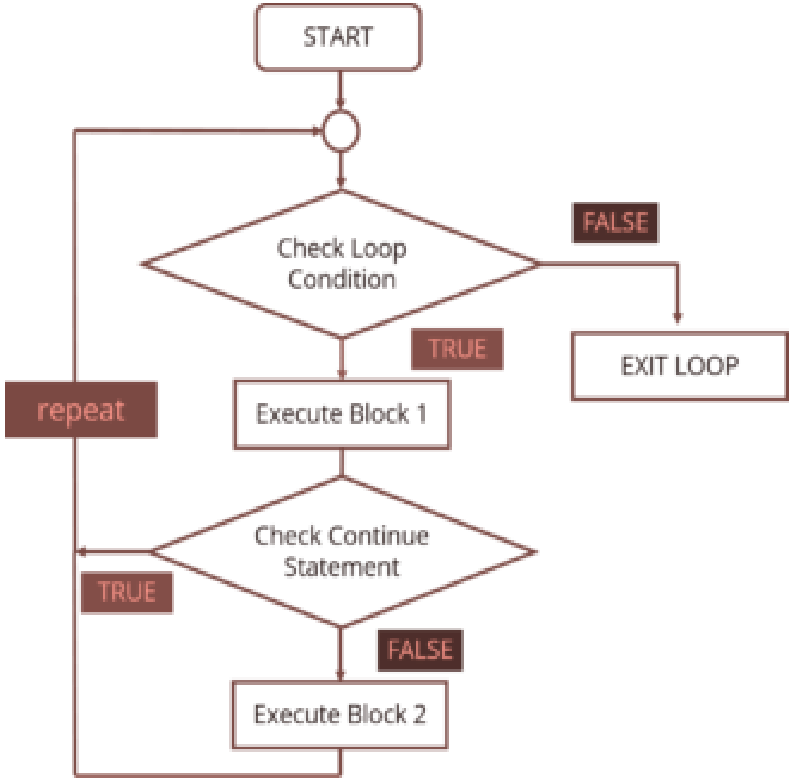


Continue statement

- **Continue statement:** Continue statement is another type of control statements. The continue keyword causes the loop to immediately jump to the next iteration of the loop.

- In this flowchart, the code will respond in the following steps:

  1. First of all, it will enter the loop where it checks the condition. If the loop condition is false, it directly exits the loop.
  2. If the loop condition is true, it will execute block 1 statements.
  3. After that it will check for 'continue' statement. If it is present, then the statements after that will not be executed in the same iteration of the loop.
  4. If 'continue' statement is not present, then all the statements after that will be executed.



Assignment

1. Write a Java program to check a number it is positive or negative.
2. Write a Java program to find greatest of 3 numbers.
3. Write a program in Java to display the multiplication table of 5.
   - Expected Output :
   - 5 X 0 = 0
   - 5 X 1 = 5
   - 5 X 2 = 10
   - ----------
   - ----------
   - 5 X 10 = 50
4. Write a Java program count the number of digits of the number.
5. Java program to reverse a number.
6. Number is Palindrome or not

Agenda

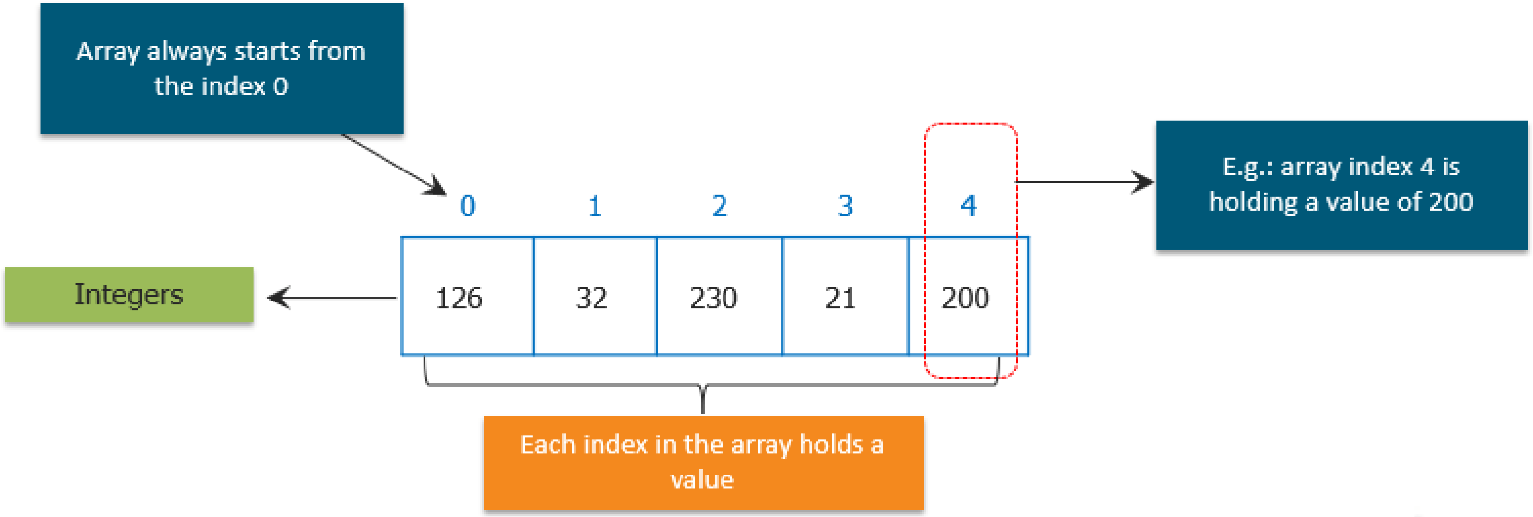
- Arrays
- Single Dimensional Array
- Multi Dimensional Array
- Strings

What are Java Arrays?

- An *array* is a container object that holds a fixed number of values of a single type.
- The length of an array is established when the array is created. After creation, its length is fixed.
- There are 2 types of arrays

Single Dimensional Array

- Declare array
- Insert values into array
- Find the size of array
- How to read/access values from array



Array always starts from the index 0

E.g.: array index 4 is holding a value of 200

Integers

Each index in the array holds a value

Multi Dimensional Array

- Declare array
- Insert values into array
- Find the size of array
- How to read/access values from array

```
int [][]a= new int [2][2];
```

	0	1
0	1	4
1	4	5

2 x 2 dimensional int array

```
char [][]a= new char[3][2];
```

	0	1
0	s	a
1	g	v
2	v	d

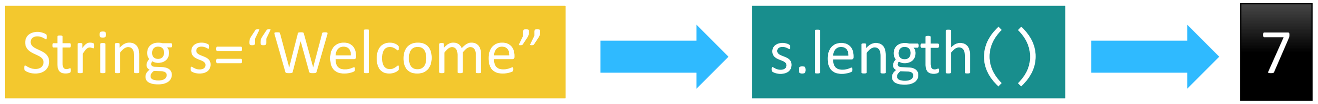
```
float [][]a= new float[5][5];
```

	0	1	2	3	4
0	2.2	3.4	5.0	3.3	1.2
1	7.8	9.0	1.1	2.9	5.5
2	2.0	3.0	7.8	9.8	9.9
3	5.7	6.6	8.8	5.3	2.7
4	1.8	4.4	7.6	1.0	1.1

5 x 5 dimensional float array

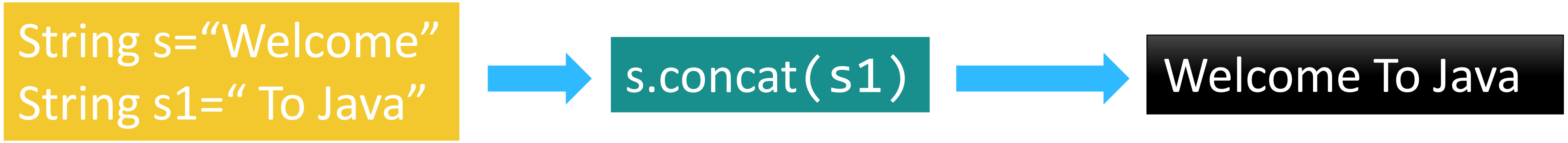
Strings

- **length():** It returns count of total number of characters present in the String.



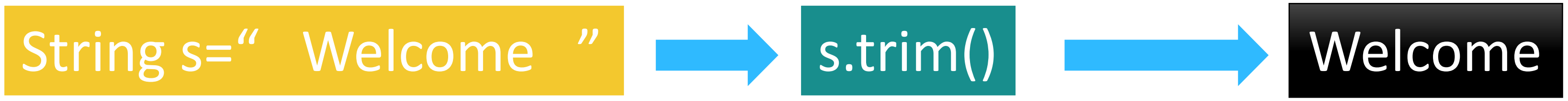
String s=“Welcome” → s.length( ) → 7

- **concat() :** Combines a specific string at the end of another string and ultimately returns a combined string. It is like appending another string.



String s=“Welcome”
String s1=“ To Java”
→ s.concat(s1) → Welcome To Java

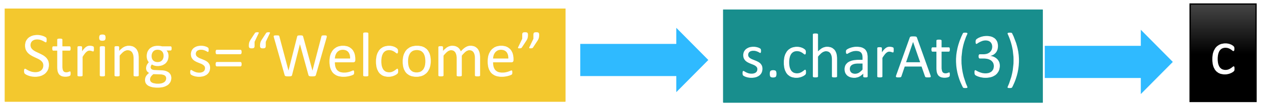
- **trim()** : The java string trim() method removes the leading and trailing spaces.



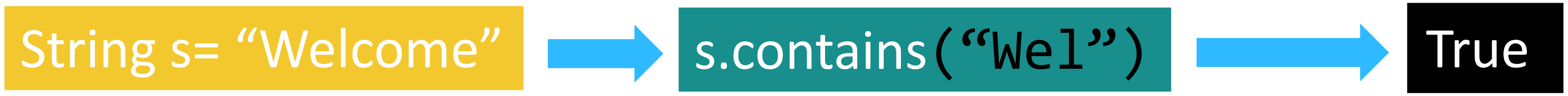
String s=“   Welcome   ” → s.trim() → Welcome

Strings

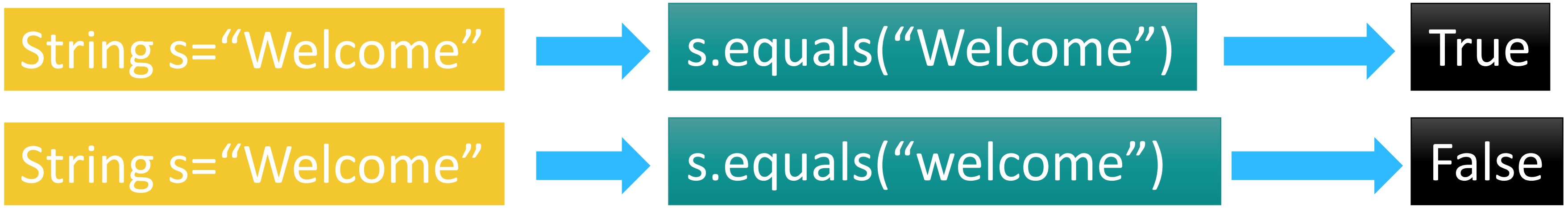
- **charAt():** Returns a char value at the given index number. The index number starts from 0.



- **contains()** : Searches the sequence of characters in this string. It returns true if sequence of char values are found in this string otherwise returns false.

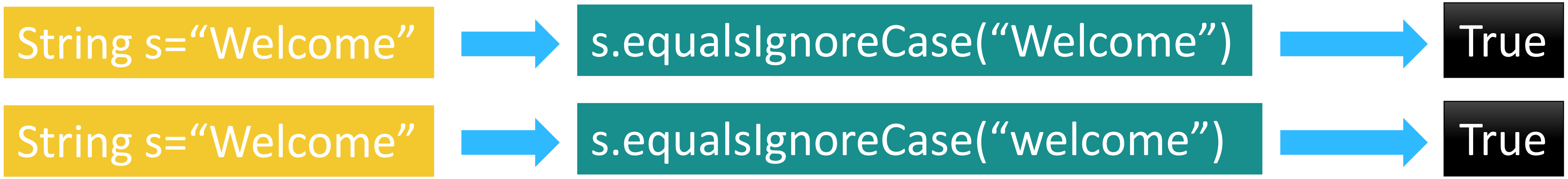


- **equals() :** Compares the two given strings based on the content of the string. If any character is not matched, it returns false. If all characters are matched, it returns true.



Strings

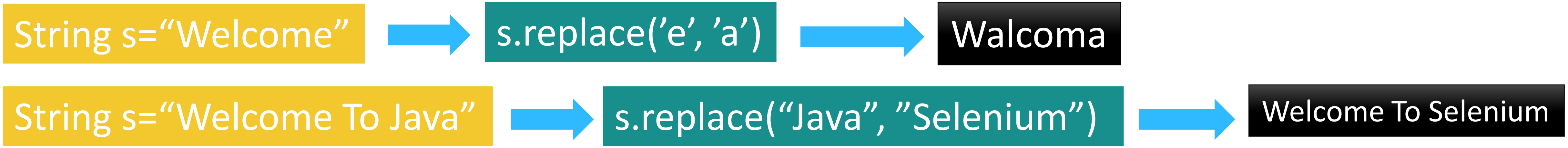
- **equalsIgnoreCase() :** Compares two string on the basis of content but it does not check the case like equals() method. In this method, if the characters match, it returns true else false.



String s="Welcome" → s.equalsIgnoreCase("Welcome") → True

String s="Welcome" → s.equalsIgnoreCase("welcome") → True

- **replace():** Returns a string, replacing all the old characters or CharSequence to new characters. There are 2 ways to replace methods.

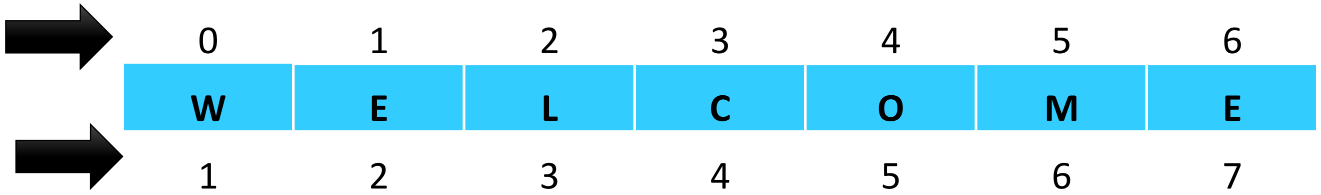


String s="Welcome" → s.replace('e', 'a') → Walcoma

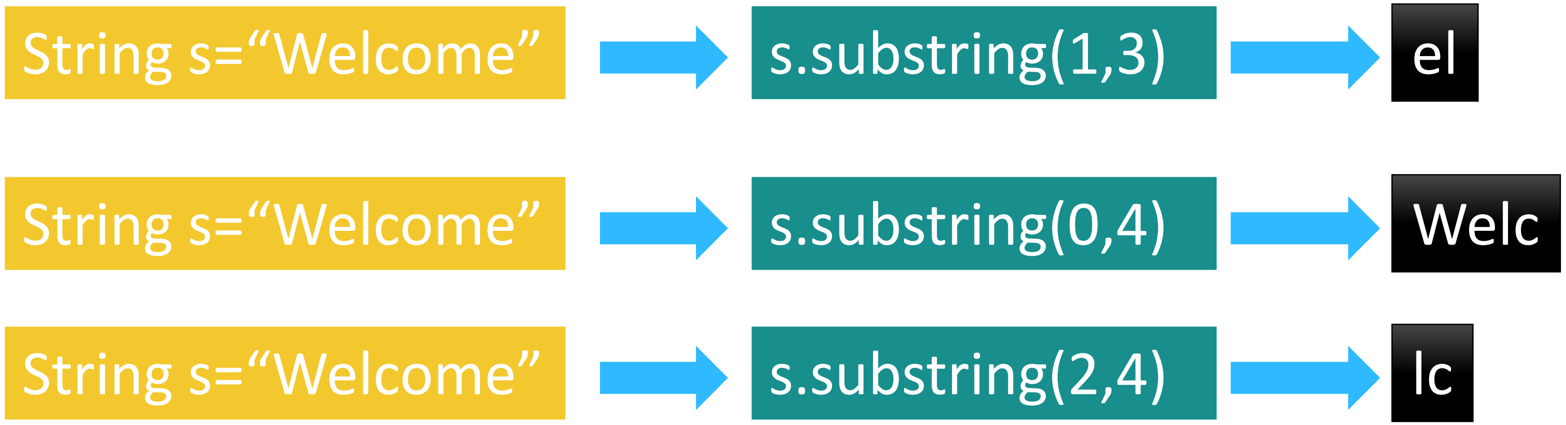
String s="Welcome To Java" → s.replace("Java", "Selenium") → Welcome To Selenium

Strings

- **Substring() :** Returns substring of a string based on starting index and ending index.



0	1	2	3	4	5	6
W	**E**	**L**	**C**	**O**	**M**	**E**
1	2	3	4	5	6	7



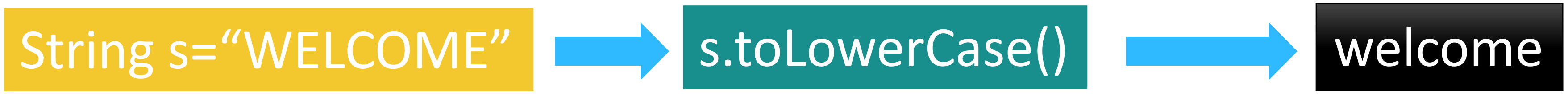
String s=“Welcome” → s.substring(1,3) → el

String s=“Welcome” → s.substring(0,4) → Welc

String s=“Welcome” → s.substring(2,4) → lc

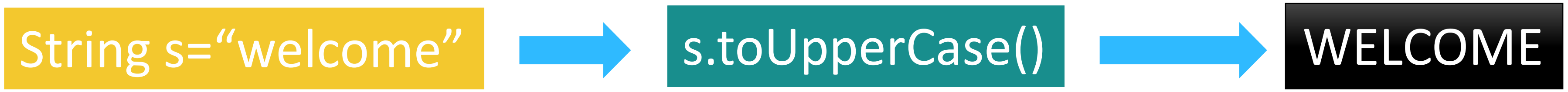
Strings

- **toLowerCase():** returns the string in lowercase letter.



String s="WELCOME" → s.toLowerCase() → welcome

- **toUpperCase():** returns the string in Uppercase letter.



String s="welcome" → s.toUpperCase() → WELCOME

Assignment (Arrays)

1. Write a Java program to calculate sum values of an array.
2. Write a Java program to search specific number in an array.
3. Define an array with some string values, write a Java program to search specific string in an array.
4. Write a Java program to print even and odd numbers in an array.
5. Write a program to find greatest and smallest element in an array.
6. Write a program to add two matrices.

Assignment (Strings)

1. Write a java program to compare two strings, ignoring case differences.
2. Write a Java program to concatenate a given string to the end of another string.
3. Write a java program to get the length of a given string
4. Write a Java program to get a substring of a given string between two specified positions
5. Write a Java program to convert all the characters in a string to uppercase.
6. Write a Java program to convert all the characters in a string to lowercase

Agenda

- Class & Object
- Java Methods
- Call By Value & Call By Reference
- Java Constructor

Java Class & Object

Class

```
class Employee
{
        int eid;
        String ename;
        double sal;
        String job;

        void display()
        {
        System.out.println(eid);
        System.out.println(ename);
        System.out.println(sal);
        System.out.println(job);
        }
        void bonus()
        {
        System.out.println((sal *10) /100);
        }
}
```

Variables

Methods

Object1

```
Employee emp1=new Employee();
emp1.eid=1020;
emp1.ename="John";
emp1.sal=80000;
emp1.job="Manager";

emp1.display();
```

Object2

```
Employee emp2=new Employee();
emp2.eid=1021;
emp2.ename="David";
emp2.sal=50000;
emp2.job="Tech Assistant";

emp2.display();
```

Class & Object

- main() within class

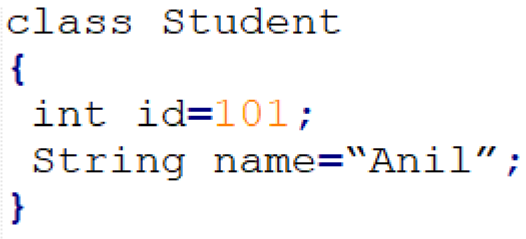
```
class Student{
 int id=101;  //field or data member or instance variable
 String name=“Anil”;

 public static void main(String args[]){
  Student s1=new Student();//creating an object of Student
  System.out.println(s1.id);//accessing member through reference variable
  System.out.println(s1.name);
 }
}
```

Class & Object

▪ main() outside class

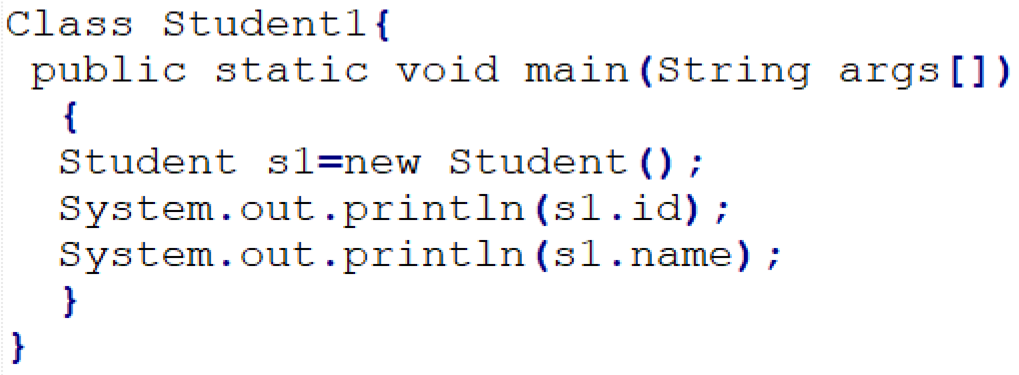
- In real time development, we create classes and use it from another class. It is a better approach than previous one.
- We can have multiple classes in different java files or single java file.

Student.java



```
class Student
{
 int id=101;
 String name="Anil";
}
```

Student1.java



```
Class Student1{
 public static void main(String args[])
  {
  Student s1=new Student();
  System.out.println(s1.id);
  System.out.println(s1.name);
  }
}
```

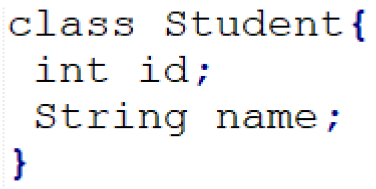
Class & Object

- 3 ways to initialize object variables in java.
  - By reference variable
  - By method
  - By constructor

Class & Object

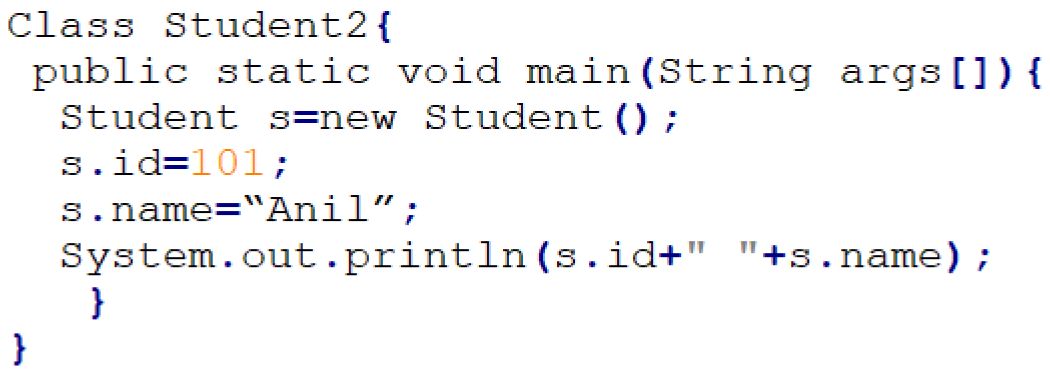
- Initialization through reference variable

Student.java



```
class Student{
 int id;
 String name;
}
```

Student2.java

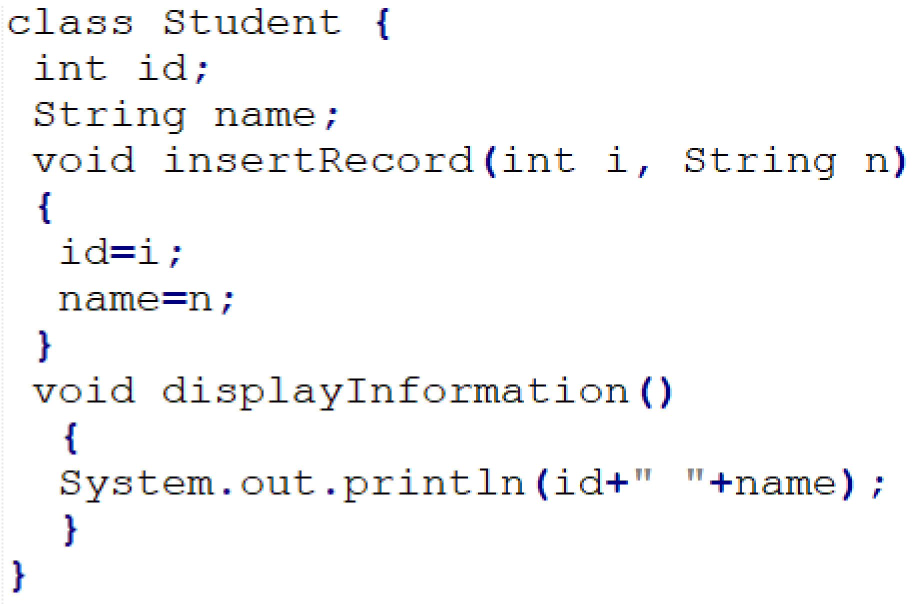


```
Class Student2{
 public static void main(String args[]){
  Student s=new Student();
  s.id=101;
  s.name=“Anil”;
  System.out.println(s.id+" "+s.name);
   }
}
```

Class & Object

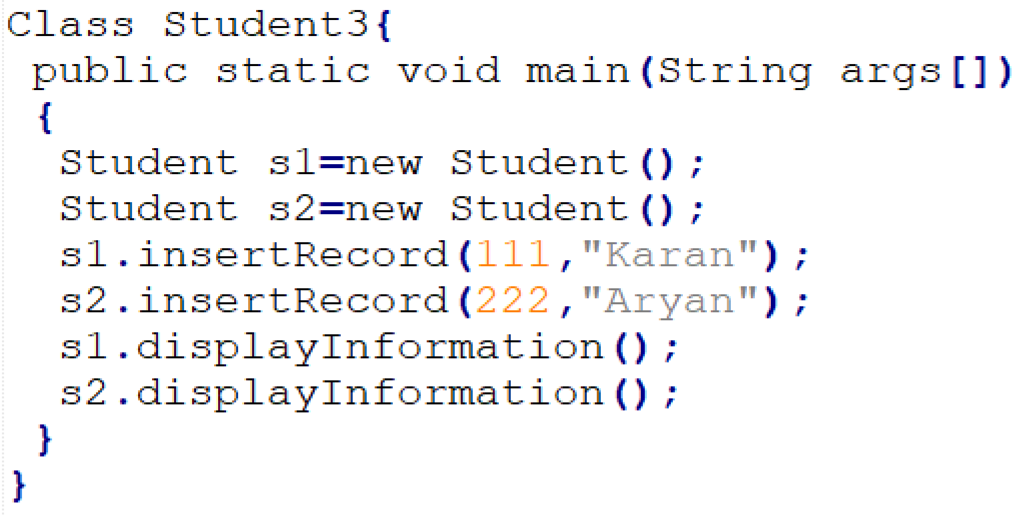
- Initialization through method

Student.java



```
class Student {
 int id;
 String name;
 void insertRecord(int i, String n)
 {
  id=i;
  name=n;
 }
 void displayInformation()
  {
  System.out.println(id+" "+name);
  }
}
```

Student3.java

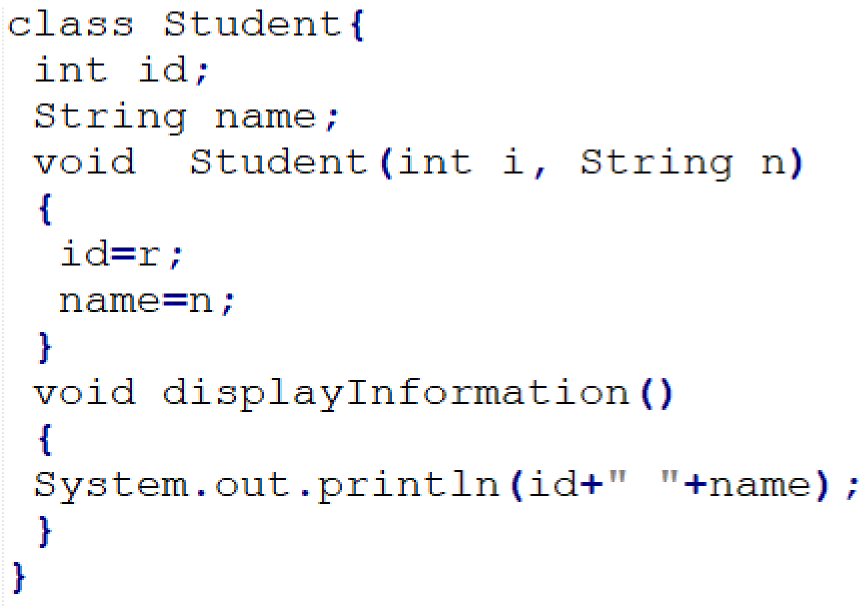


```
Class Student3{
 public static void main(String args[])
 {
  Student s1=new Student();
  Student s2=new Student();
  s1.insertRecord(111,"Karan");
  s2.insertRecord(222,"Aryan");
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Class & Object

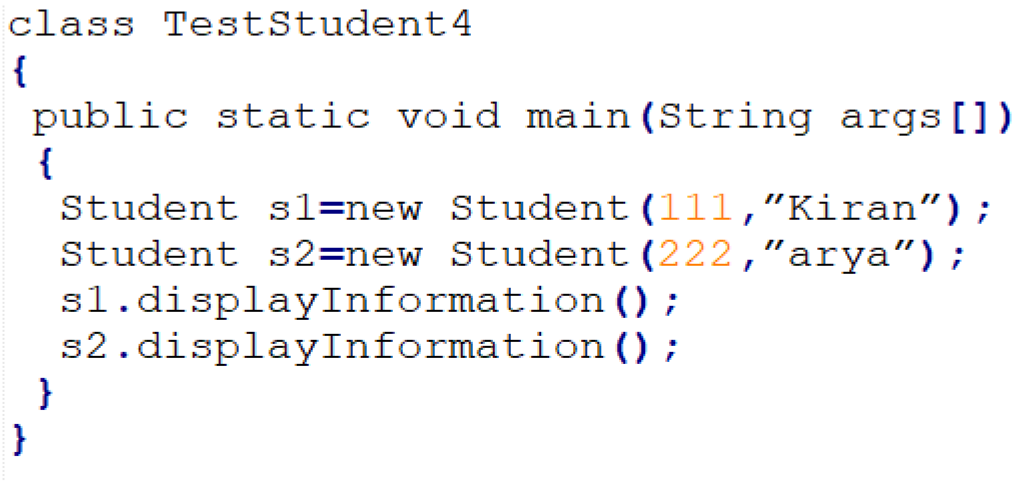
- Initialization through constructor

Student.java



```
class Student{
 int id;
 String name;
 void  Student(int i, String n)
 {
  id=r;
  name=n;
 }
 void displayInformation()
 {
 System.out.println(id+" "+name);
 }
}
```

Student4.java



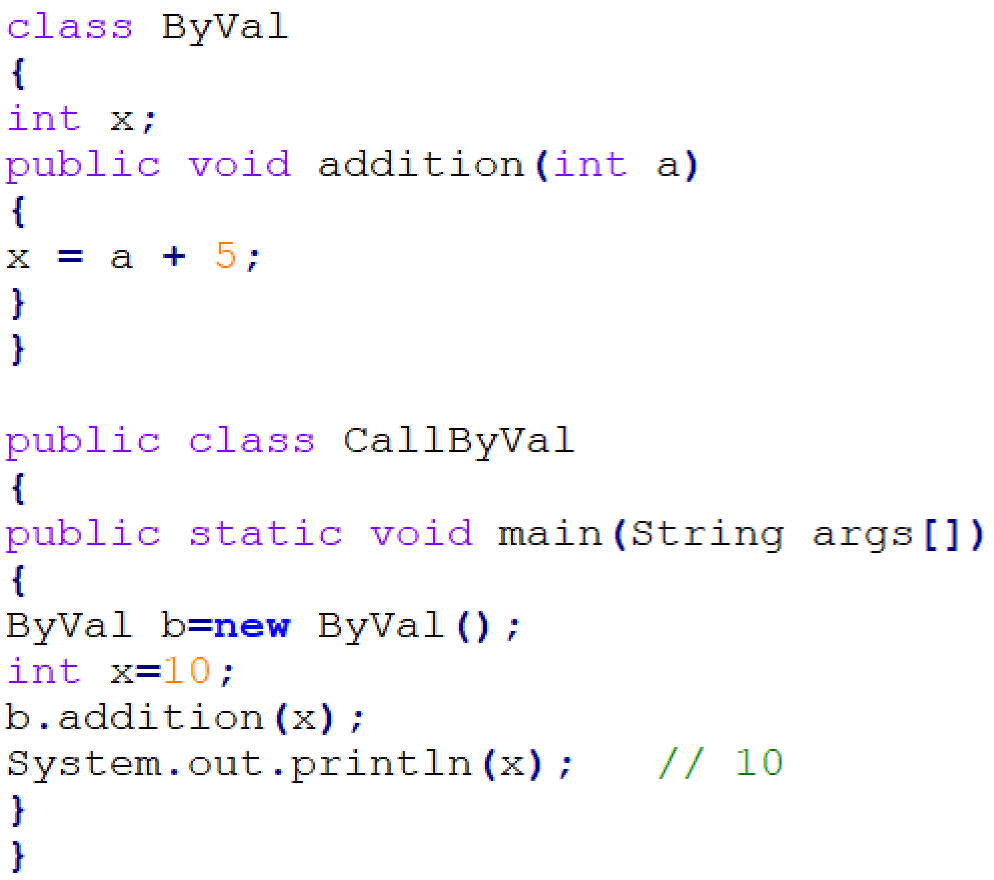
```
class TestStudent4
{
 public static void main(String args[])
 {
  Student s1=new Student(111,”Kiran”);
  Student s2=new Student(222,”arya”);
  s1.displayInformation();
  s2.displayInformation();
 }
}
```

Java Methods

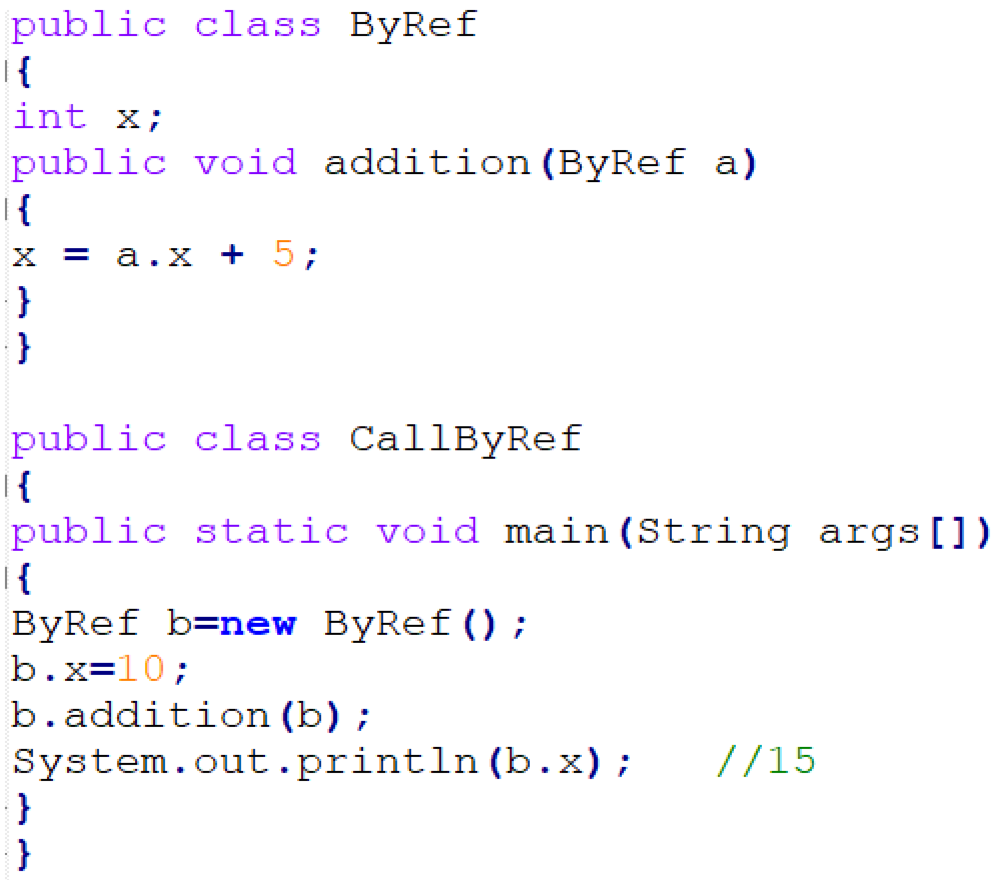
- A **method** is a set of code which is referred to by name and can be called (invoked) at any point in a program simply by utilizing the **method's** name.
- A **method** as a subprogram that acts on data and often returns a value.
- Each **method** has its own name.

	Parameter/s	Returned Value
Case1	✗	✗
Case2	✗	✓
Case3	✓	✗
Case4	✓	✓

Call by Value & Call by Reference



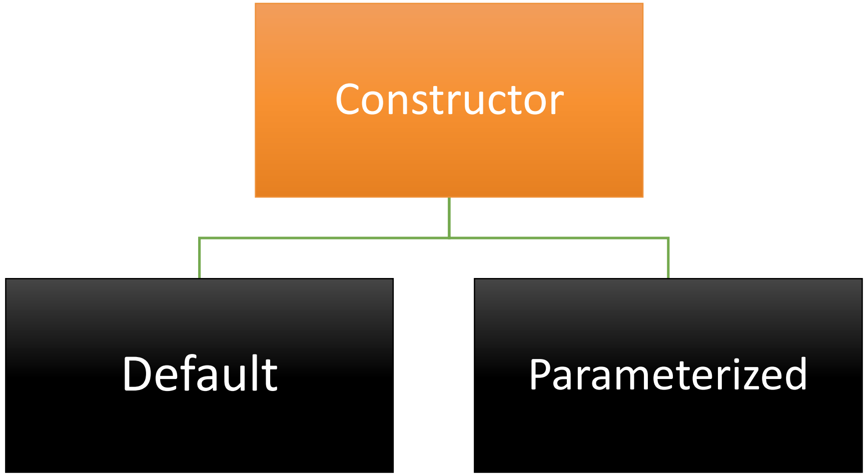
```
class ByVal
{
int x;
public void addition(int a)
{
x = a + 5;
}
}

public class CallByVal
{
public static void main(String args[])
{
ByVal b=new ByVal();
int x=10;
b.addition(x);
System.out.println(x);    // 10
}
}
```



```
public class ByRef
{
int x;
public void addition(ByRef a)
{
x = a.x + 5;
}
}

public class CallByRef
{
public static void main(String args[])
{
ByRef b=new ByRef();
b.x=10;
b.addition(b);
System.out.println(b.x);   //15
}
}
```

Java Constructor

- **Constructor** in java is a special type of method that is used to initialize the object.
- Java constructor is invoked at the time of object creation.
- **Rules for creating java constructor:**
  1. Constructor name must be same as its class name.
  2. Constructor must have no explicit return type.
- There are 2 Types of Constructors.



Method V/s Constructor

Method

- Method name can be anything.
- Method can return a value.
- Need to call method explicitly.

Constructor

- Constructor name must be same as class name.
- Constructor doesn't return a value.
- Automatically invoked at the time of object creation.

Assignment

1. **Create a Student class contains the following variables and methods.**
   - **Class Name**: Student
   - **Variables** : SID , Sname, Sub1,Sub2,Sub3
   - **Methods:**
     - getStuData() Takes student details SID and Sname as parameters and assign them to variables.
     - getStuMarks() Takes student marks as parameters and assign them to Sub1, Sub2, Sub3.
     - totalMarks() Calculate total marks and print the student details with total marks.
   - Now, create objects from Student class stu1, stu2 etc. Then call Student class methods.
2. **Write a program to demonstrate constructor.**
   - Create a class 'Calculation' with 3 integer variable.
   - Create a constructor for assign the values into variables.
   - Then create another method 'sum' to calculate sum of 3 numbers.
   - Now, create object and call constructor by passing 3 integer values then call sum method.

Agenda

- Overloading
- this keyword
- Static variables & static methods

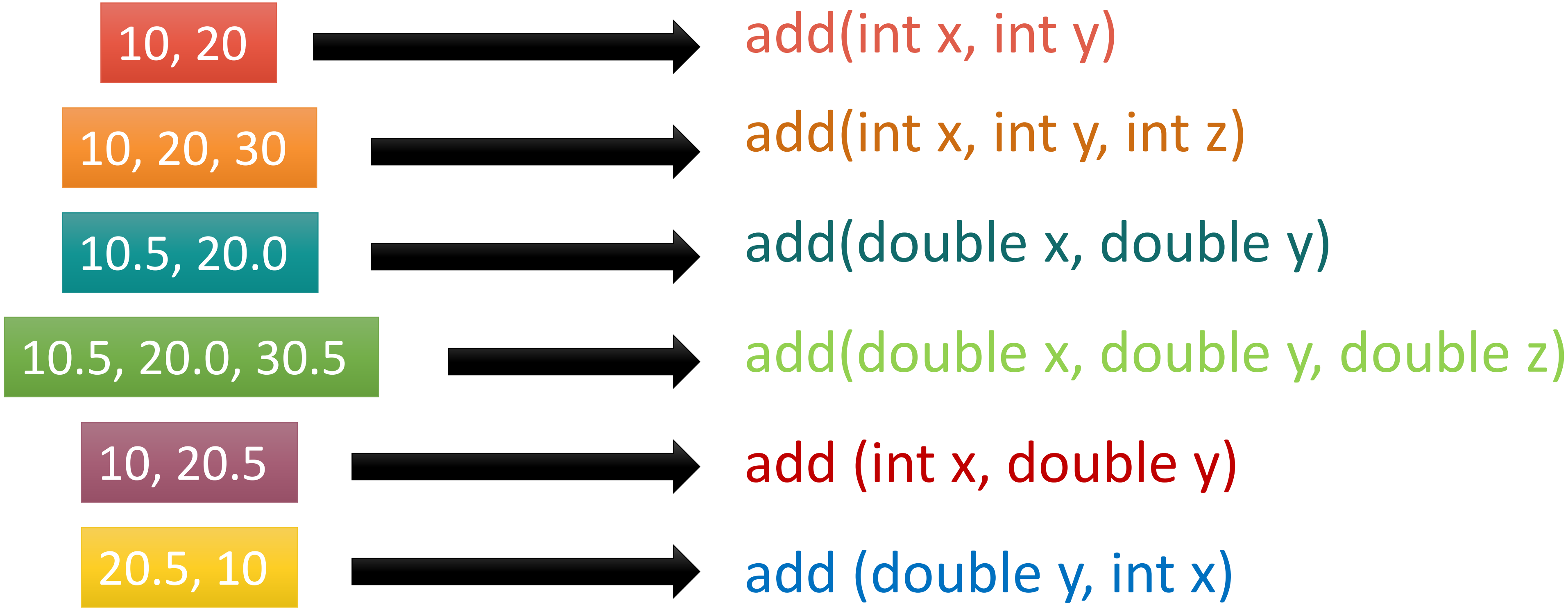
Method Overloading

- Method Overloading in Java is a concept related to Object Oriented Programming (OOP). Java supports overloading of methods and can distinguish between different methods with method signatures. A situation, wherein, in the same class there are two or more methods with same name, having different functions or different parameters, it is called Method Overloading.

- **Why Method Overloading?**
  - Using Method Overloading in Java is very common among Java programmers, because it:
  - Provides flexibility to call similar method for different data types
  - Saves memory
  - Saves time

- **Method Overloading can be done in two ways:**
  - ➢ By changing Arguments’ data types
  - ➢ By changing number of Arguments

Overloading



10, 20 → add(int x, int y)

10, 20, 30 → add(int x, int y, int z)

10.5, 20.0 → add(double x, double y)

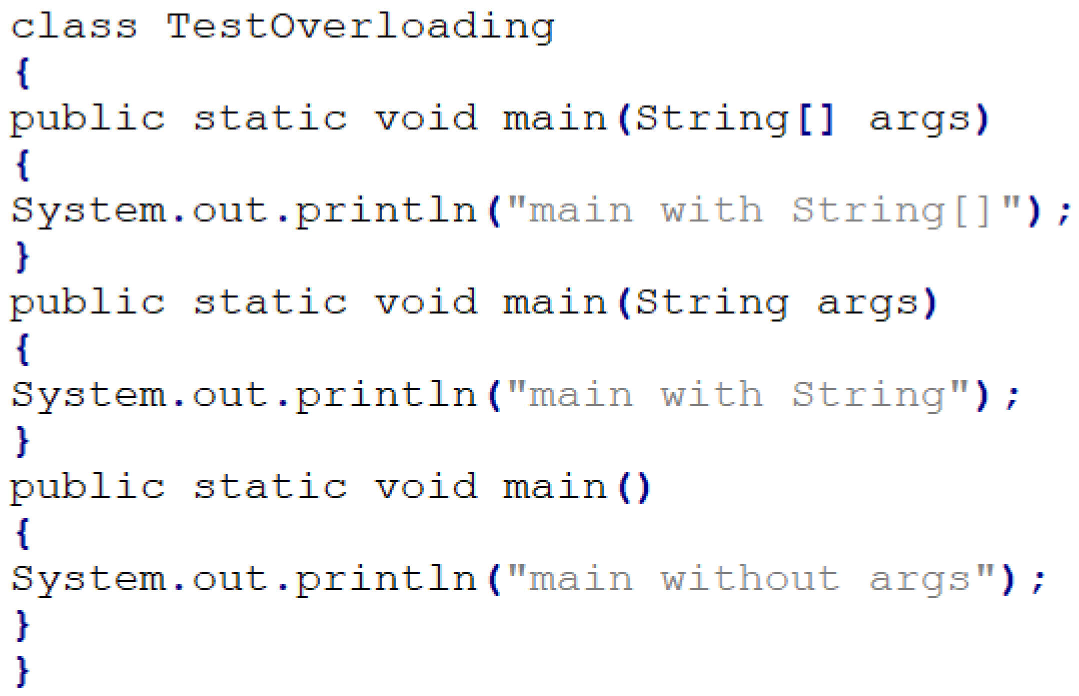
10.5, 20.0, 30.5 → add(double x, double y, double z)

10, 20.5 → add (int x, double y)

20.5, 10 → add (double y, int x)

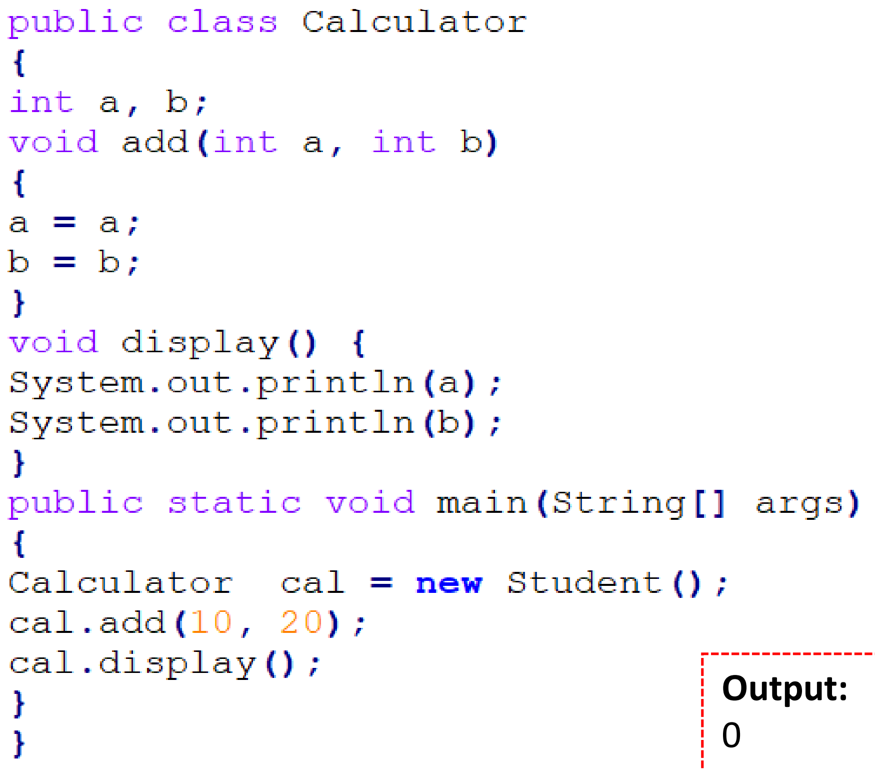
Can we overload java main() method?

- **Yes,** by method overloading. You can have any number of main methods in a class by method overloading. But JVM calls main() method which receives string array as arguments only.



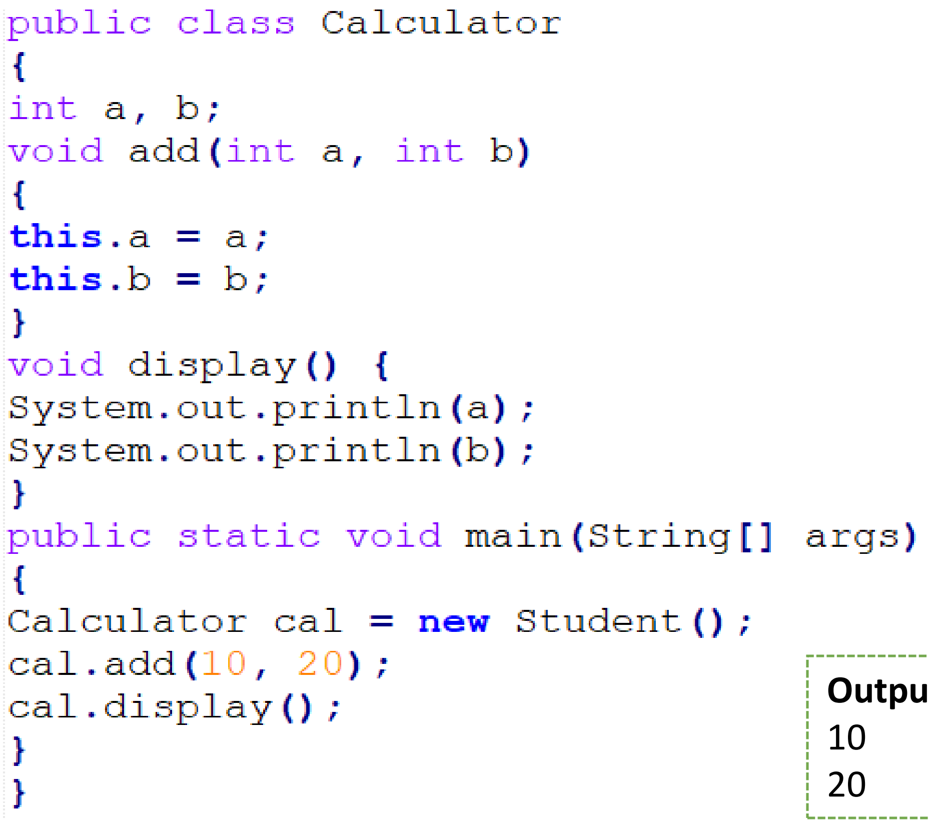
```
class TestOverloading
{
public static void main(String[] args)
{
System.out.println("main with String[]");
}
public static void main(String args)
{
System.out.println("main with String");
}
public static void main()
{
System.out.println("main without args");
}
}
```

this keyword



```
public class Calculator
{
int a, b;
void add(int a, int b)
{
a = a;
b = b;
}
void display() {
System.out.println(a);
System.out.println(b);
}
public static void main(String[] args)
{
Calculator  cal = new Student();
cal.add(10, 20);
cal.display();
}
}
```

Output:
0
0

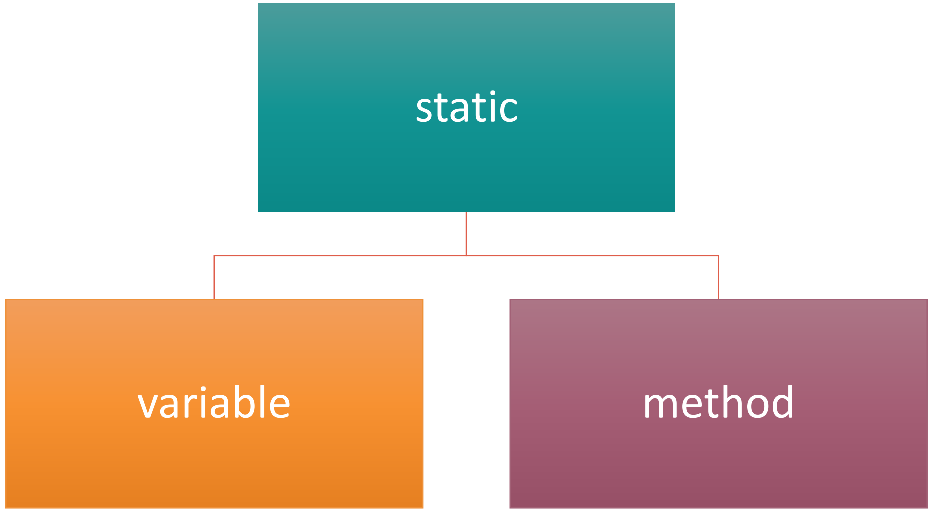


```
public class Calculator
{
int a, b;
void add(int a, int b)
{
this.a = a;
this.b = b;
}
void display() {
System.out.println(a);
System.out.println(b);
}
public static void main(String[] args)
{
Calculator cal = new Student();
cal.add(10, 20);
cal.display();
}
}
```

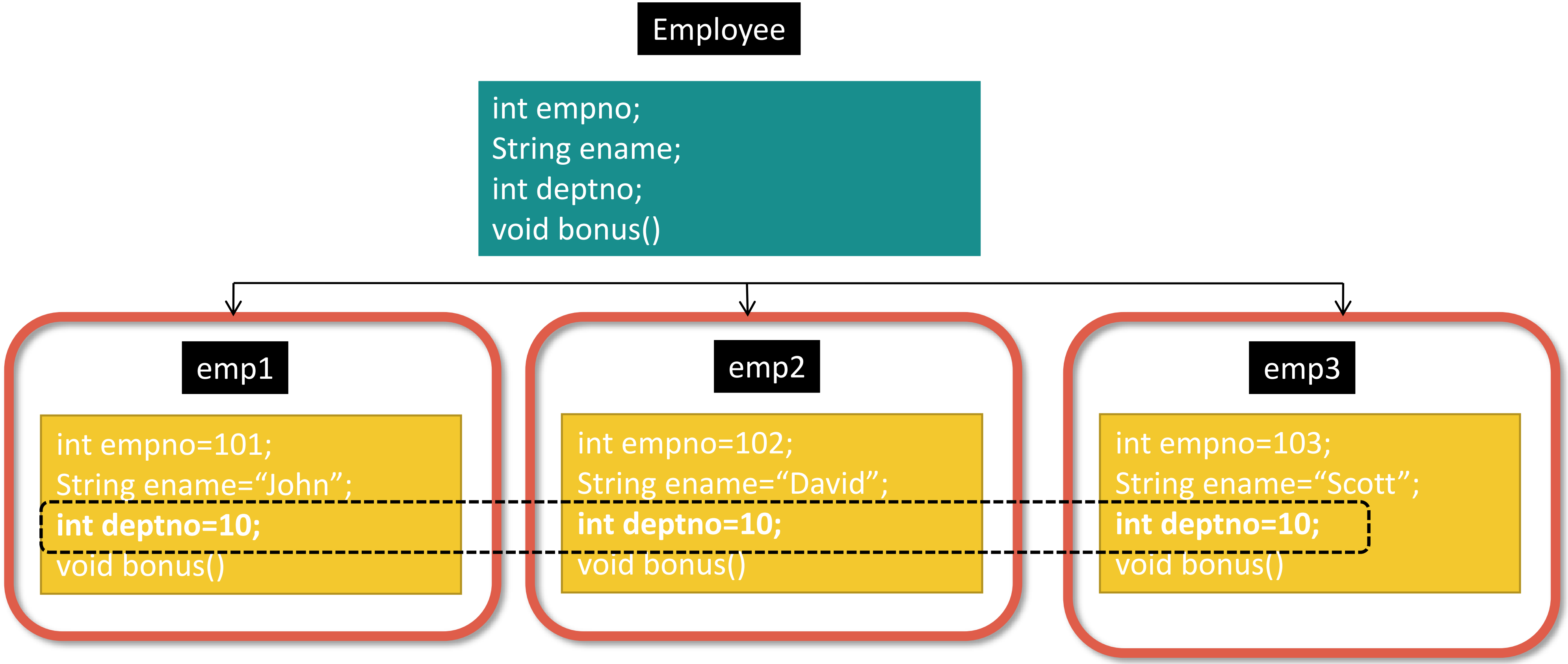
Output:
10
20

static

- The **static keyword** in java is used for memory management mainly.
- We can apply java static keyword with variables and methods
- The static can be:
  - variable (also known as class variable)
  - method (also known as class method)

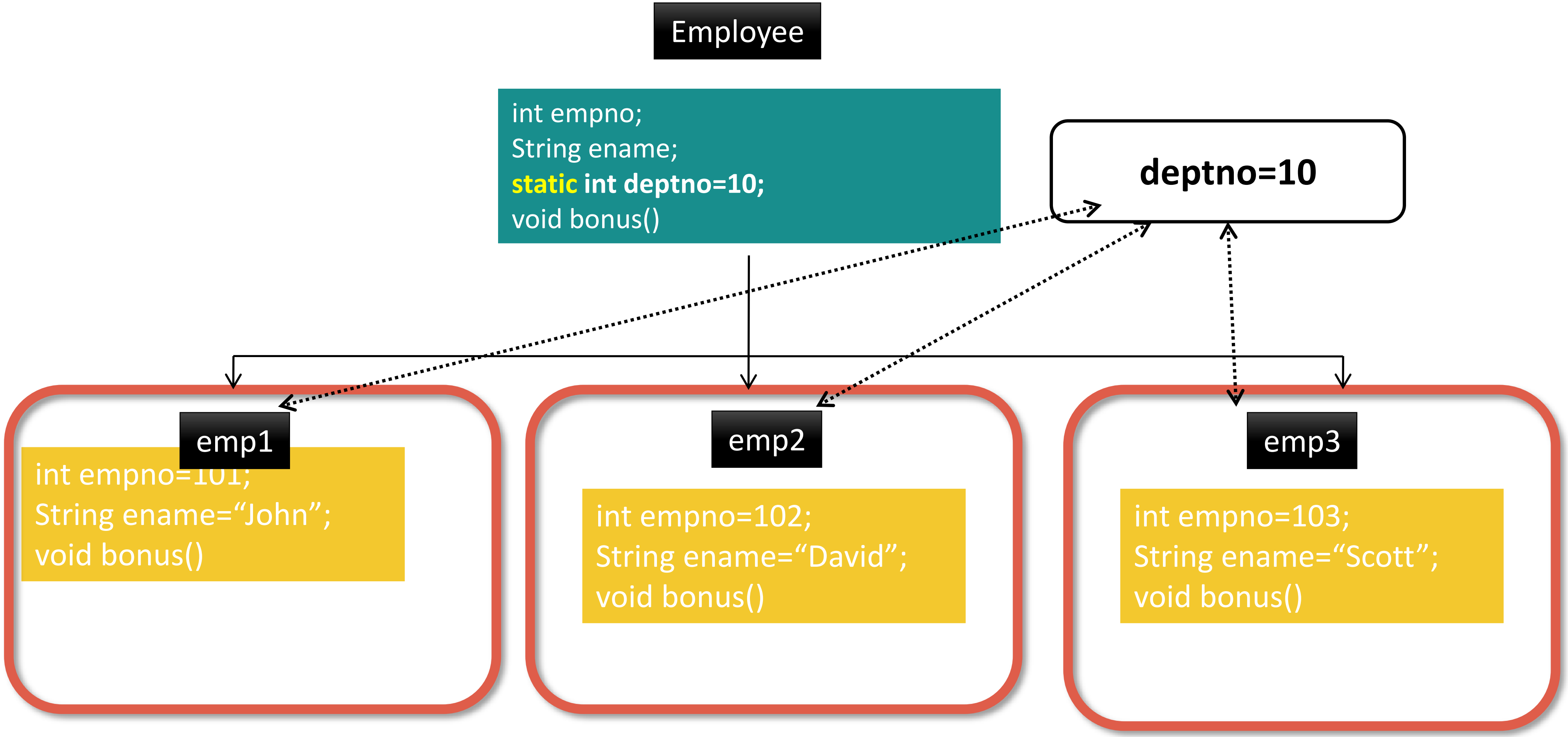


static



- Every object occupy certain amount of space in memory.
- Objects are independent. So if modify a variable value in obj1 , that will not reflect other objects variable values.

static



Employee

```
int empno;
String ename;
static int deptno=10;
void bonus()
```

deptno=10

emp1

```
int empno=101;
String ename=“John”;
void bonus()
```

emp2

```
int empno=102;
String ename=“David”;
void bonus()
```

emp3

```
int empno=103;
String ename=“Scott”;
void bonus()
```

static variables and methods

	static		Non-static	
	Variables	Methods	Variables	Methods
static methods	✓	✓	✓	✓
Non-static methods	✓	✓	✓	✓

✓ Direct Access

✓ Through Object

System.out.println()

```
class Test
{
static String s="Welcome";
}
```

Test.s.length();

```
class System
{
static PrintStream out;
}
```

System.out.println();

Assignment

- **1) Create a class Calculation with the following methods.**
- **Class Name:** Calculation

➢ int sum(int x, int y) : Should accept two integer parameters and returns sum of two numbers.

➢ int sum(int x, int y, int z) : Should accept three integer parameters and returns sum of three numbers.

➢ double sum(double x, double y) : Should accept two double type parameters and returns sum of two numbers.

➢ double sum(double x, double y, double z) : Should accept three double type parameters and returns sum of three numbers.

- Now, create object for Calculations class 'cal' then call different methods by passing different inputs.

Agenda

- Java Inheritance
- Method Overriding
- super Keyword
- final Keyword

Inheritance

- In OOP, computer programs are designed in such a way where everything is an object that interact with one another. Inheritance is one such concept where the properties of one class can be inherited by the other.

- It helps to reuse the code and establish a relationship between different classes.

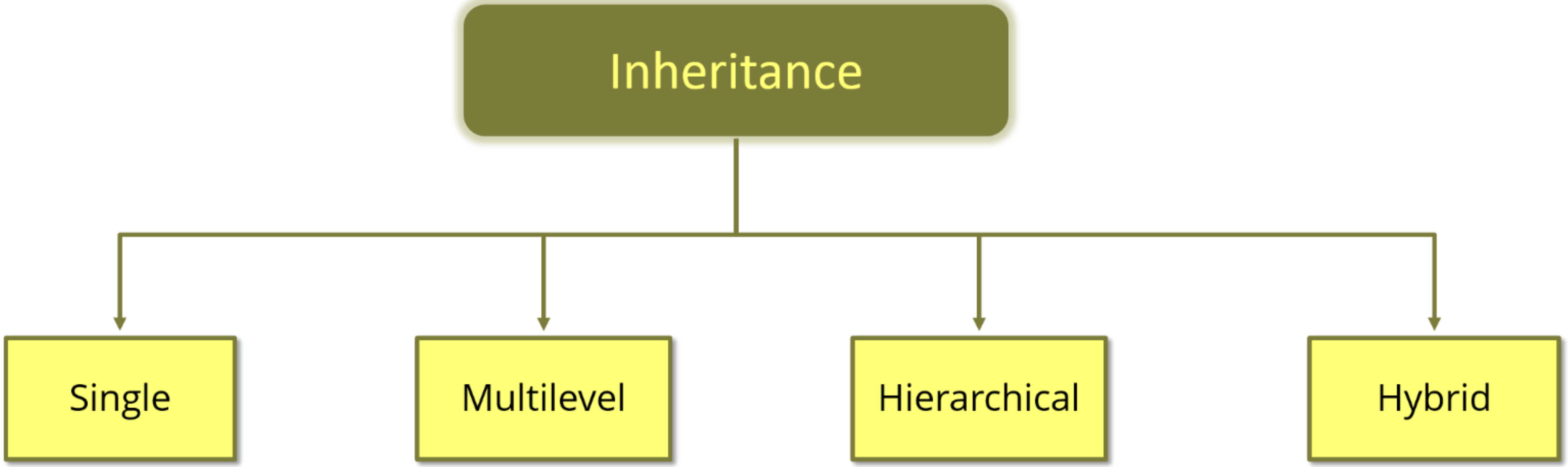
- In Java, there are two classes:

  1. Parent class (Super or Base class)
  2. Child class (Subclass or Derived class)

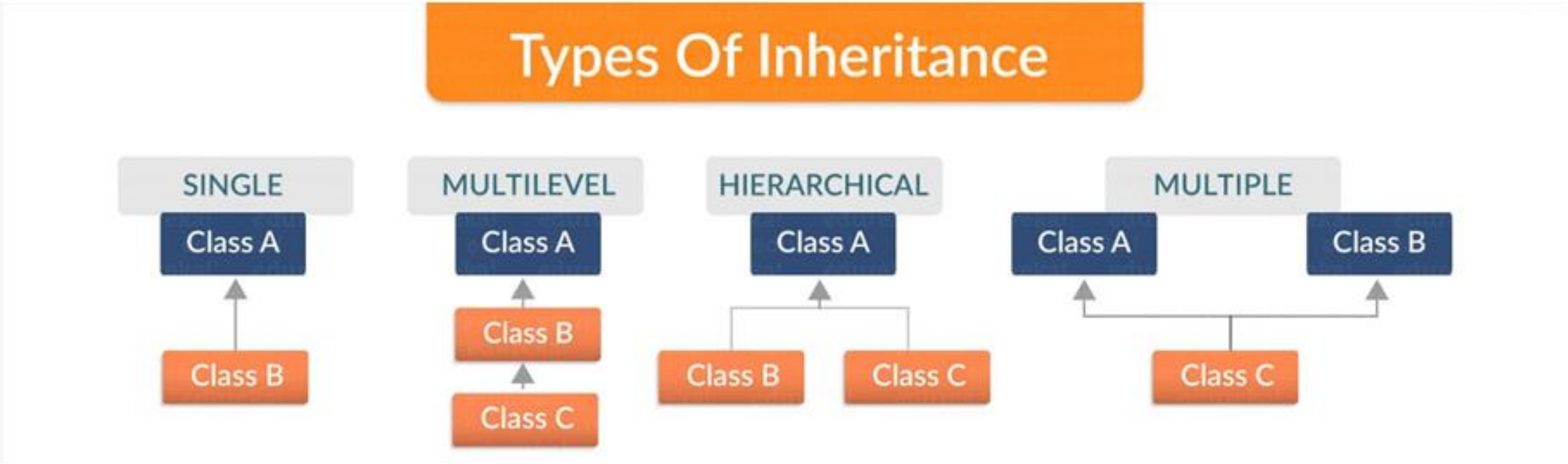
- A class which inherits the properties is known as Child Class whereas a class whose properties are inherited is known as Parent class.

Types of Inheritance

- Inheritance is further classified into 4 types.

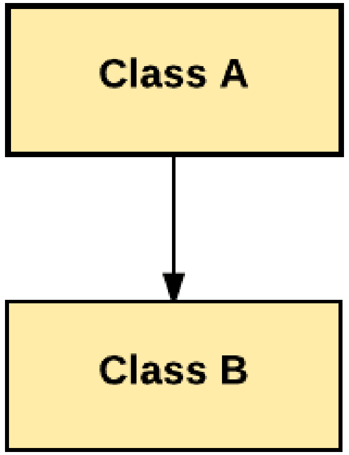


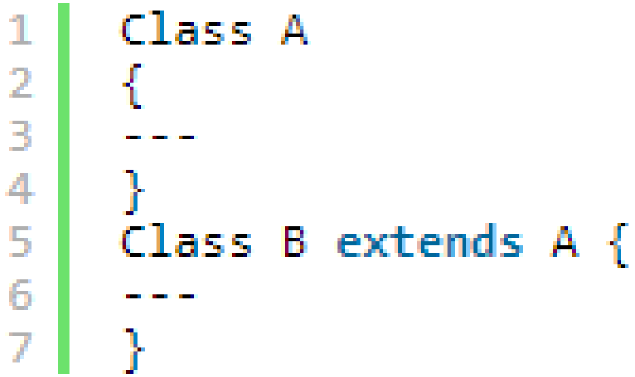
Types of Inheritance..



Single Inheritance

- In single inheritance, one class inherits the properties of another. It enables a derived class to inherit the properties and behavior from a single parent class.
- This will in turn enable code reusability as well as add new features to the existing code.
- Here, Class A is your parent class and Class B is your child class which inherits the properties and behavior of the parent class.



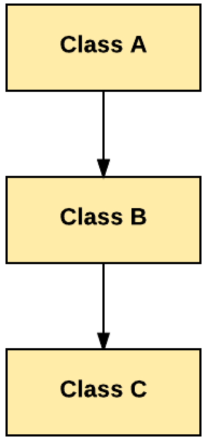


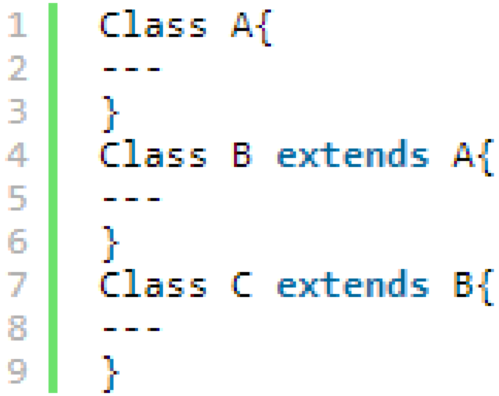
```
Class A
{
---
}
Class B extends A {
---
}
```

Multilevel Inheritance

- When a class is derived from a class which is also derived from another class, i.e. a class having more than one parent class but at different levels, such type of inheritance is called Multilevel Inheritance.

- If we talk about the flowchart, class B inherits the properties and behavior of class A and class C inherits the properties of class B. Here A is the parent class for B and class B is the parent class for C. So in this case class C implicitly inherits the properties and methods of class A along with Class B. That’s what is multilevel inheritance.



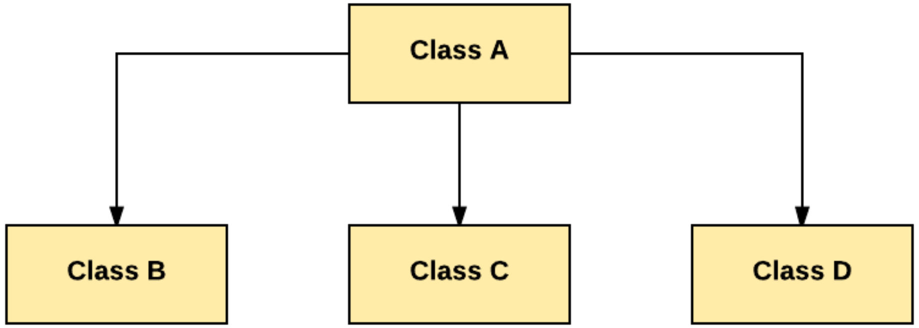


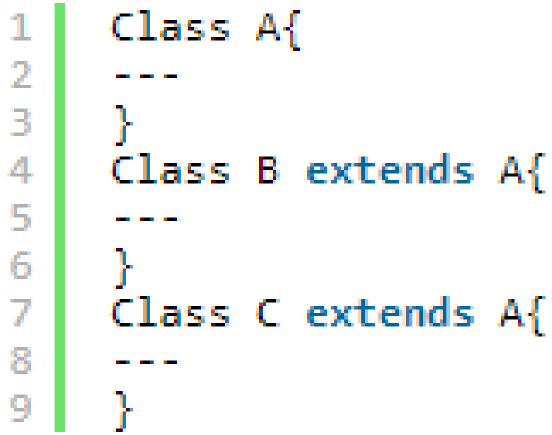
```
Class A{
---
}
Class B extends A{
---
}
Class C extends B{
---
}
```

Hierarchical Inheritance

- When a class has more than one child classes (sub classes) or in other words, more than one child classes have the same parent class, then such kind of inheritance is known as **hierarchical**.

- If we talk about the flowchart, Class B and C are the child classes which are inheriting from the parent class i.e Class A.



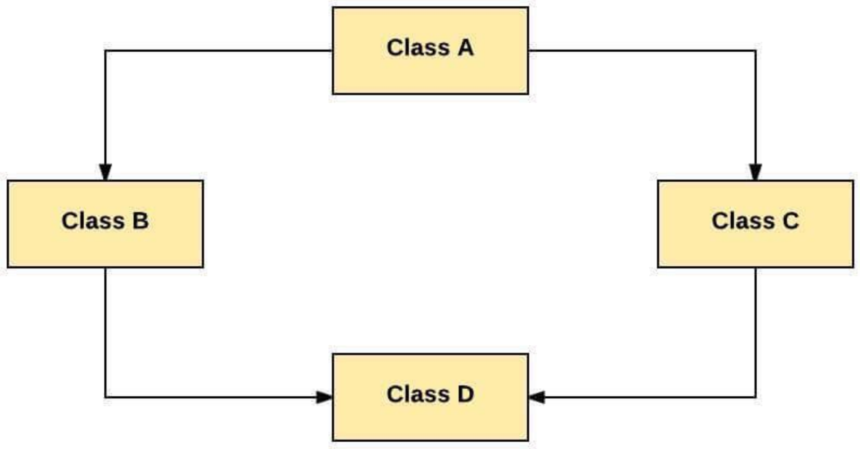


```
Class A{
---
}
Class B extends A{
---
}
Class C extends A{
---
}
```

Hybrid Inheritance

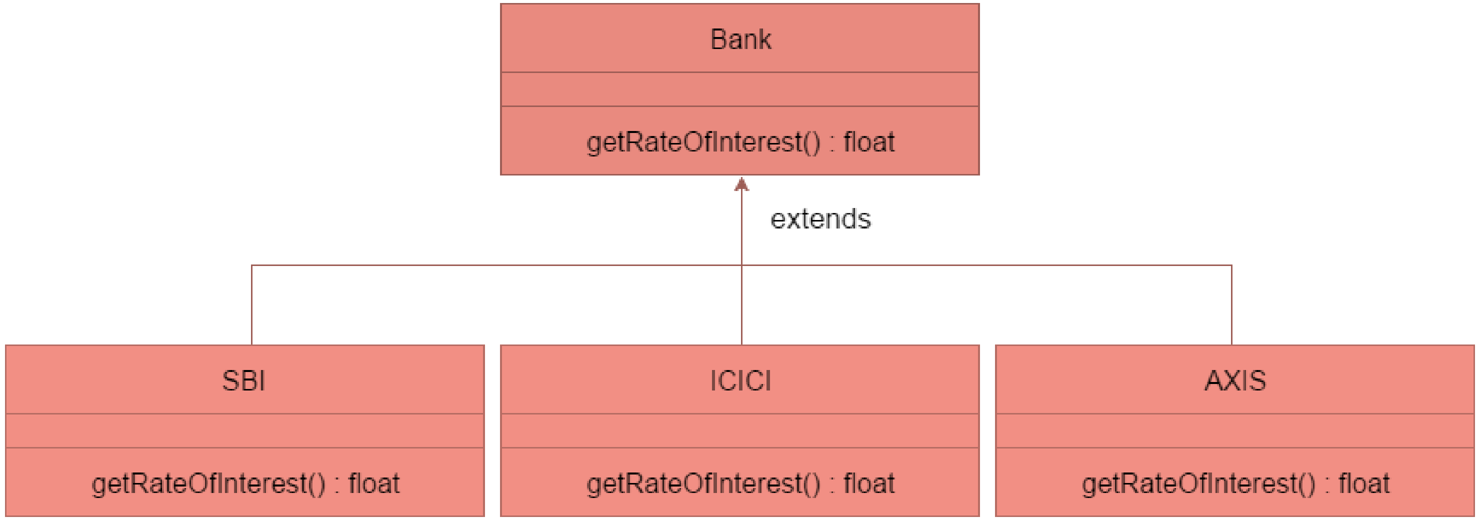
- Hybrid inheritance is a combination of *multiple* inheritance and *multilevel* inheritance. Since multiple inheritance is not supported in Java as it leads to ambiguity, so this type of inheritance can only be achieved through the use of the interfaces.

- If we talk about the flowchart, class A is a parent class for class B and C, whereas Class B and C are the parent class of D which is the only child class of B and C.



Method Overriding

- If subclass (child class) has the same method as declared in the parent class, it is known as **method overriding in java**.
- **Rules for Java Method Overriding**
  - method must have same name as in the parent class
  - method must have same parameter as in the parent class.


Bank
getRateOfInterest() : float
extends
SBI
getRateOfInterest() : float
ICICI
getRateOfInterest() : float
AXIS
getRateOfInterest() : float

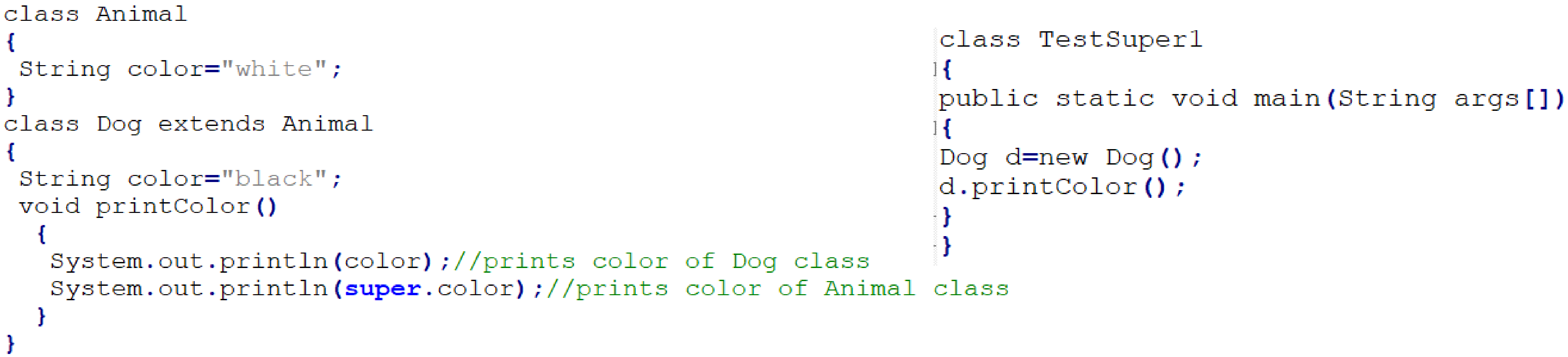
super keyword

- The **super** keyword in java is a reference variable which is used to refer immediate parent class object.

- **Usage of java super Keyword**
  - ➢ super can be used to refer immediate parent class instance variable.
  - ➢ super can be used to invoke immediate parent class method.
  - ➢ super() can be used to invoke immediate parent class constructor.

Super Keyword

- super is used to refer immediate parent class instance variable.



```
class Animal
{
 String color="white";
}
class Dog extends Animal
{
 String color="black";
 void printColor()
  {
   System.out.println(color);//prints color of Dog class
   System.out.println(super.color);//prints color of Animal class
  }
}
```

```
class TestSuper1
{
public static void main(String args[])
{
Dog d=new Dog();
d.printColor();
}
}
```

Super Keyword

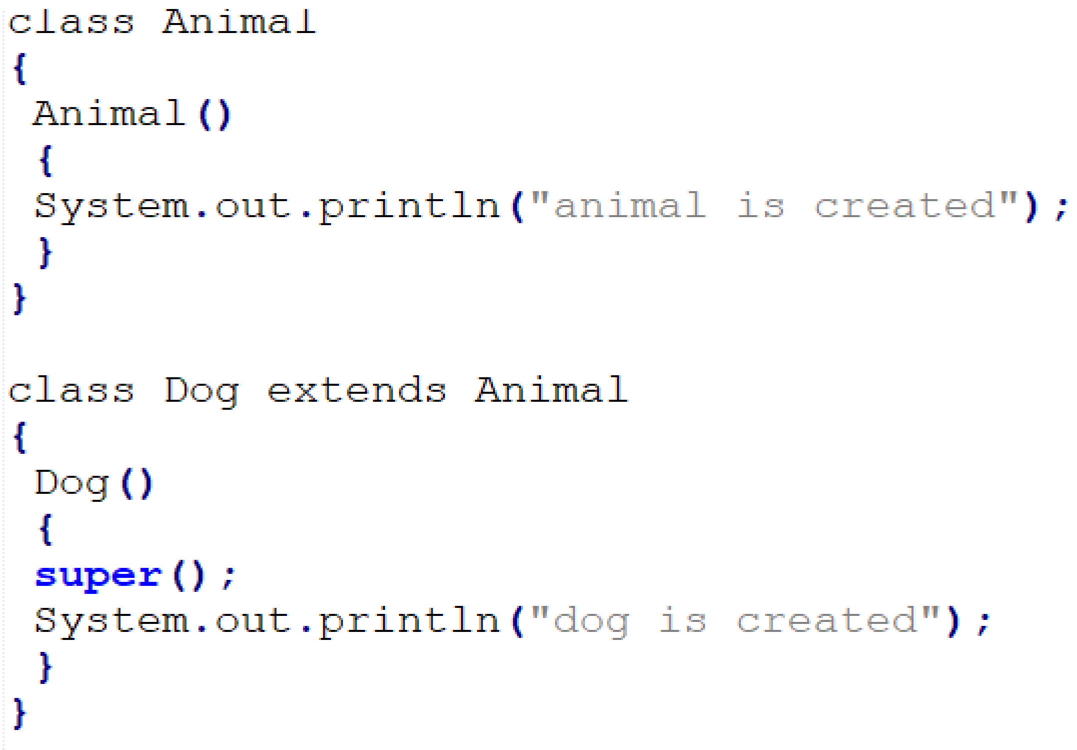
- super can be used to invoke parent class method.

```
class Animal
{
void eat()
{
System.out.println("eating...");
}
}
class Dog extends Animal
{
void eat()
{
System.out.println("eating bread...");
}

void work()
{
super.eat();
eat();
}
}
```

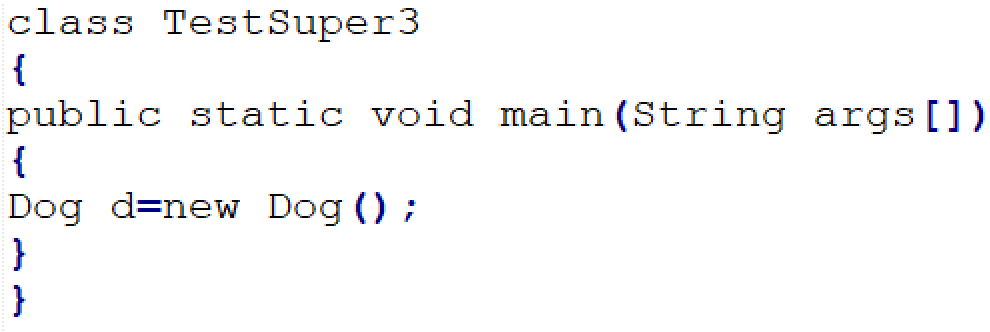
```
class TestSuper2
{
public static void main(String args[])
{
Dog d=new Dog();
d.work();
}
}
```

Super Keyword

- super is used to invoke parent class constructor



```
class Animal
{
 Animal()
 {
 System.out.println("animal is created");
 }
}

class Dog extends Animal
{
 Dog()
 {
 super();
 System.out.println("dog is created");
 }
}
```



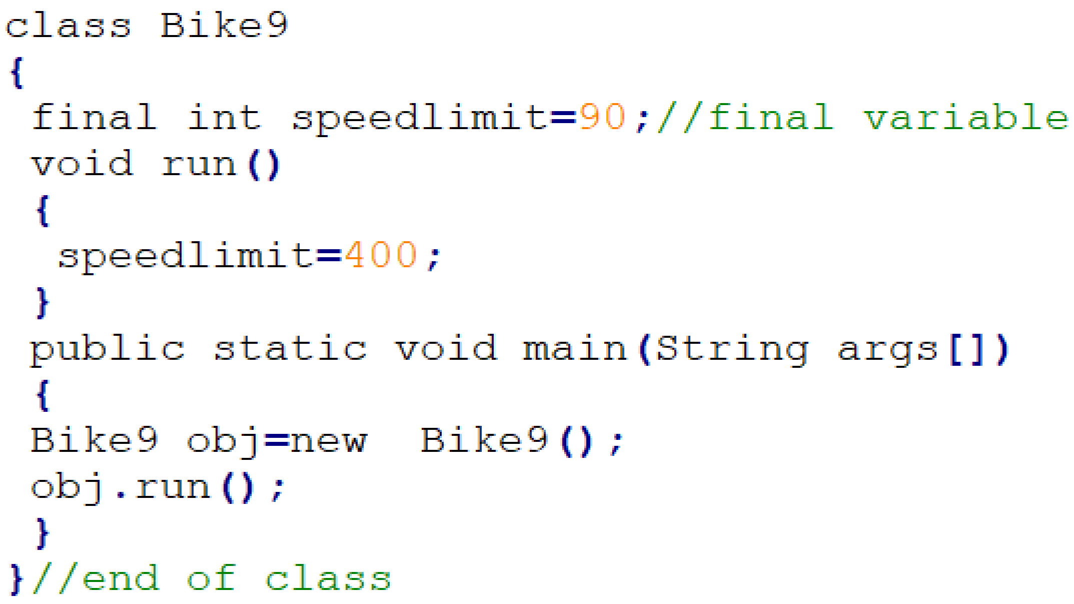
```
class TestSuper3
{
public static void main(String args[])
{
Dog d=new Dog();
}
}
```

final Keyword

- The **final keyword** in java is used to restrict the user. The java final keyword can be used for variables, methods and classes.
  - variable
  - method
  - class

Java final variable

- If you make any variable as final, you cannot change the value of final variable(It will be constant).

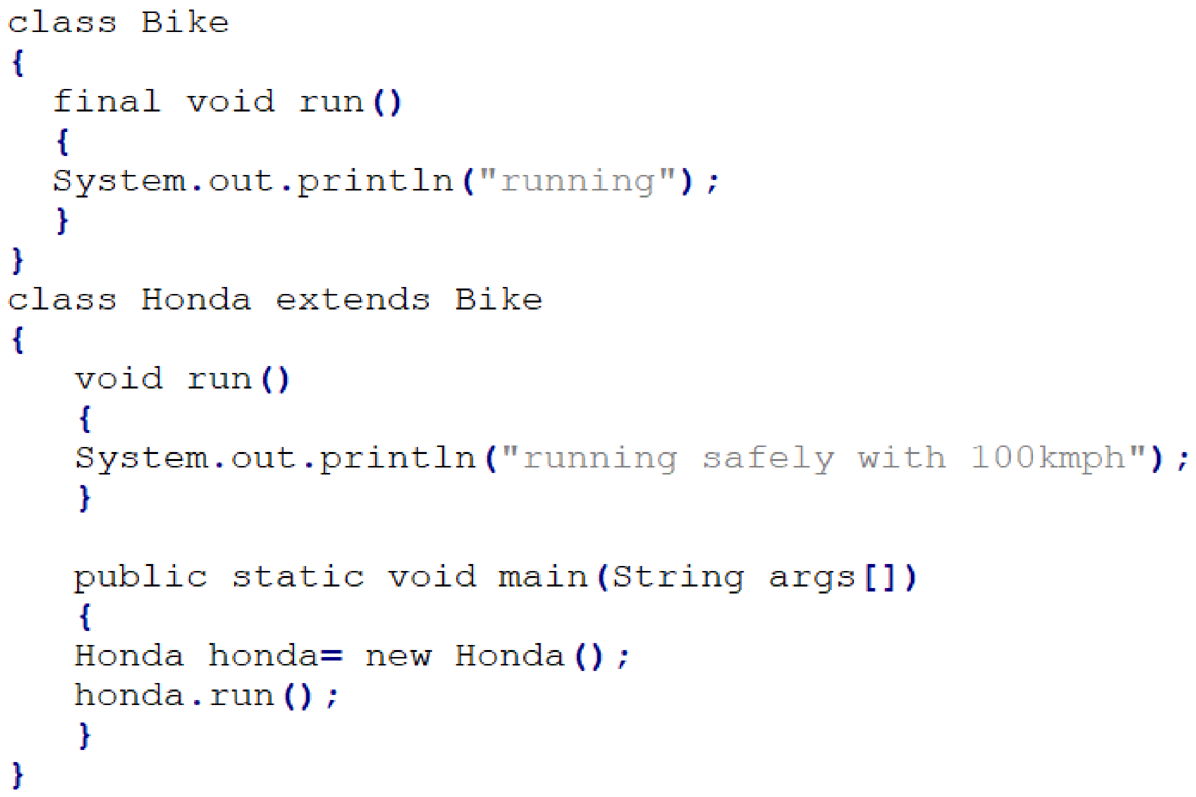


```
class Bike9
{
 final int speedlimit=90;//final variable
 void run()
 {
  speedlimit=400;
 }
 public static void main(String args[])
 {
 Bike9 obj=new  Bike9();
 obj.run();
 }
}//end of class
```

Output:Compile Time Error

Java final method

- If you make any method as final, you cannot override it.

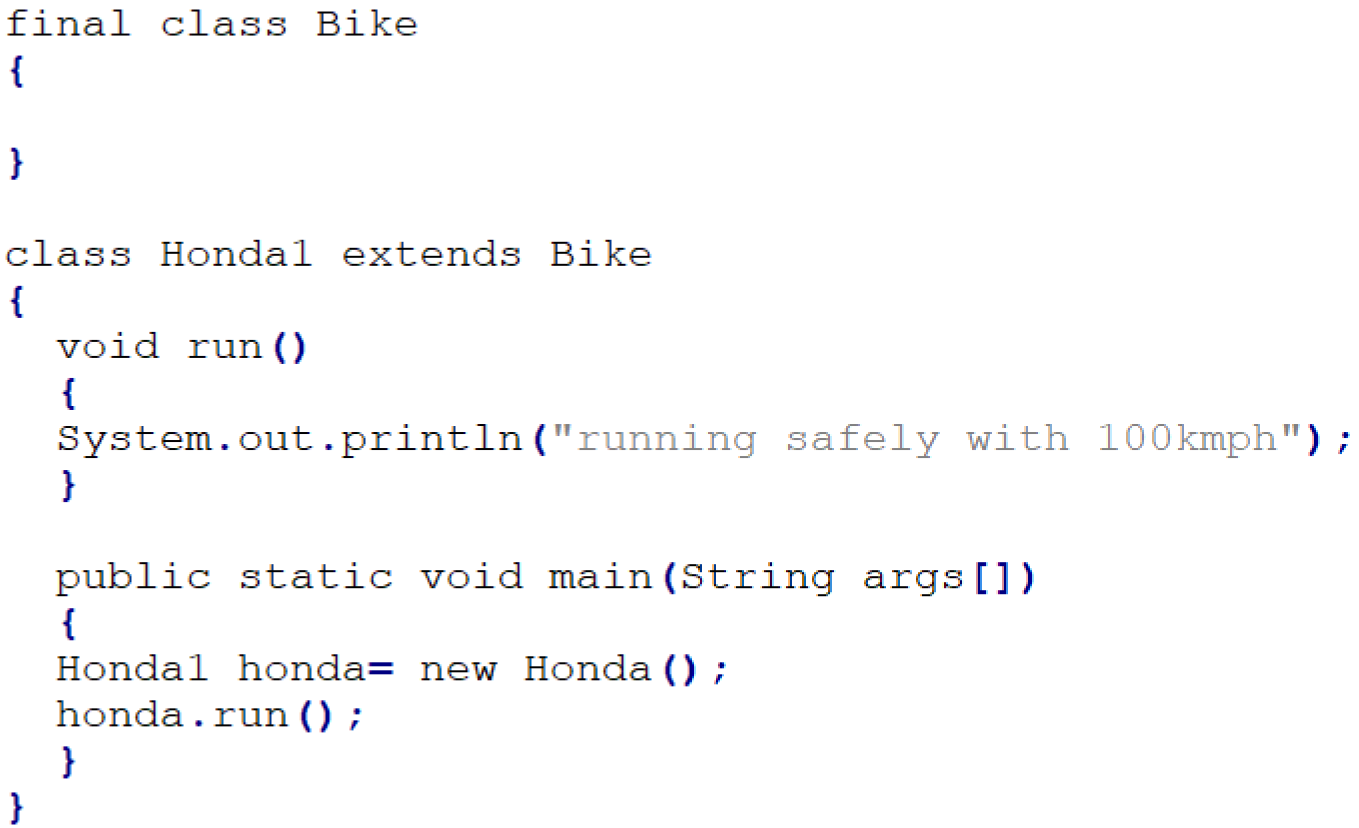


```
class Bike
{
  final void run()
  {
  System.out.println("running");
  }
}
class Honda extends Bike
{
   void run()
   {
   System.out.println("running safely with 100kmph");
   }

   public static void main(String args[])
   {
   Honda honda= new Honda();
   honda.run();
   }
}
```

Output:Compile Time Error

Java final class

- If you make any class as final, you cannot extend it.



```
final class Bike
{

}

class Honda1 extends Bike
{
  void run()
  {
  System.out.println("running safely with 100kmph");
  }

  public static void main(String args[])
  {
  Honda1 honda= new Honda();
  honda.run();
  }
}
```

Output:Compile Time Error

Assignment

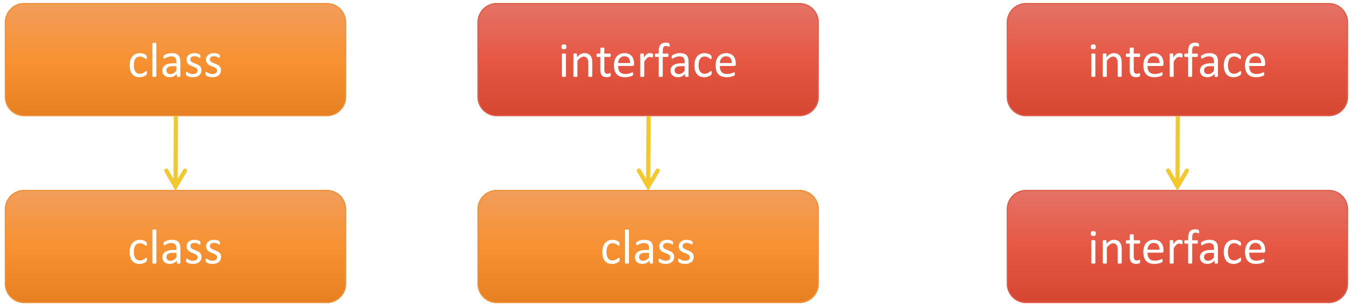
- **Assignment-1**
- Create a class ‘Teacher’ which contains following variables and methods
  - designation = "Teacher";
  - collegeName = "BusyQA";
  - does() → Teaching
- Create another class ‘ComputerTeacher’ which extends ‘Teacher’ class then create objects then call methods.

Agenda

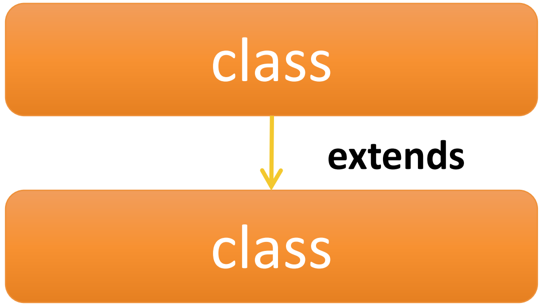
- Java Interfaces
- Java Packages
- Access Modifier’s

Java Interface

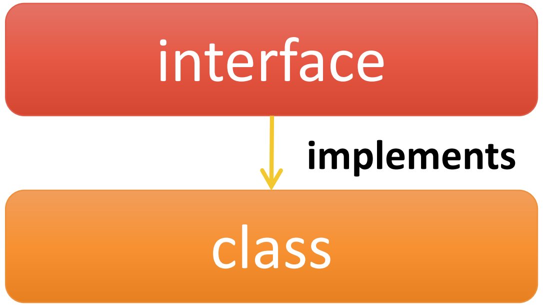
- An **interface in java** is a blueprint of a class.
- Interface contains **final and static variables**.
- Interface contains **abstract methods**.
- An **abstract method** is a method contains definition but not body.
- Methods in interface are public by default.
- Interface supports the functionality of multiple inheritance.
- We can define interface with ***interface*** keyword.
- A class extends another class, an interface extends another interface but a **class implements an interface**.
- We can create Object reference for Interface but we cannot instantiate interface.



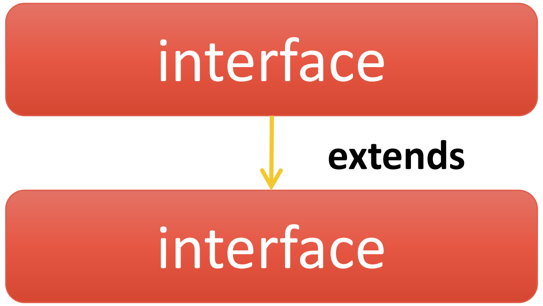
Java Interface



```
class A
{
//Variables
//Methods
}
class B extends A
{
//Variables
//Methods
}
```

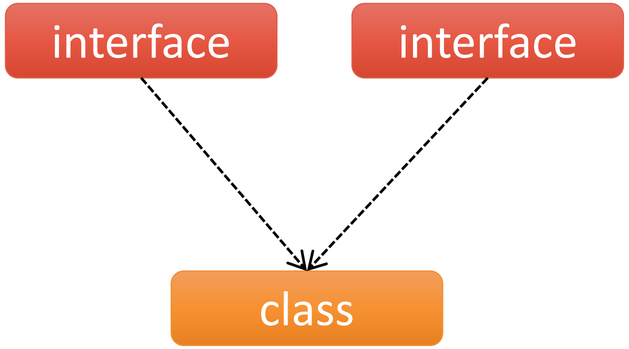


```
interface I
{
//abstract methods
//final static variables
}
class B implements I
{
//Method implementation;
}
```

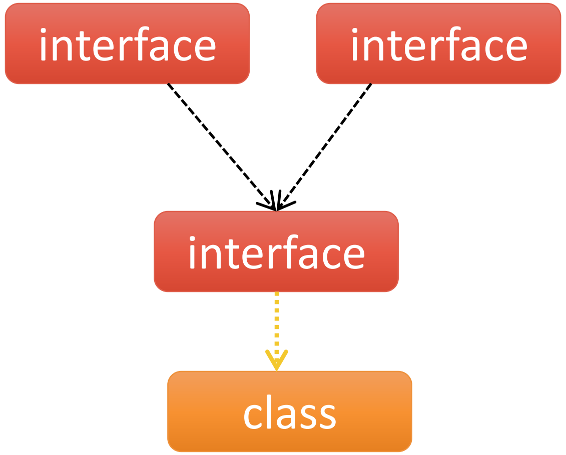


```
interface I1
{
//abstract methods
//final static variables
}
Interface I2 extends I1
{
//abstract methods
//final static variables
}
class B implements I2
{
//Methods implementation;
}
```

Multiple Inheritance in Java by Interface

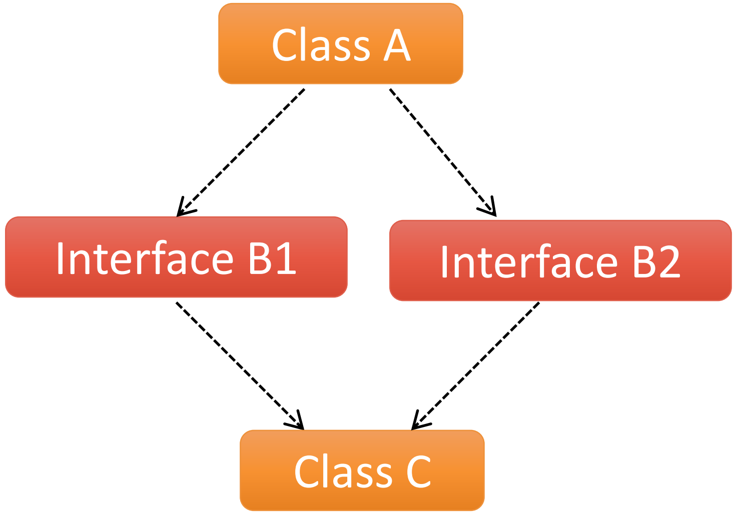


```
class A implements I1, I2
{
//Implement all the methods from I1 & I2
}
```



```
interface I  extends I1, I2
{
//final static variables
//abstract methods
}
class A implements I
{
//Implement all the methods from I
}
```

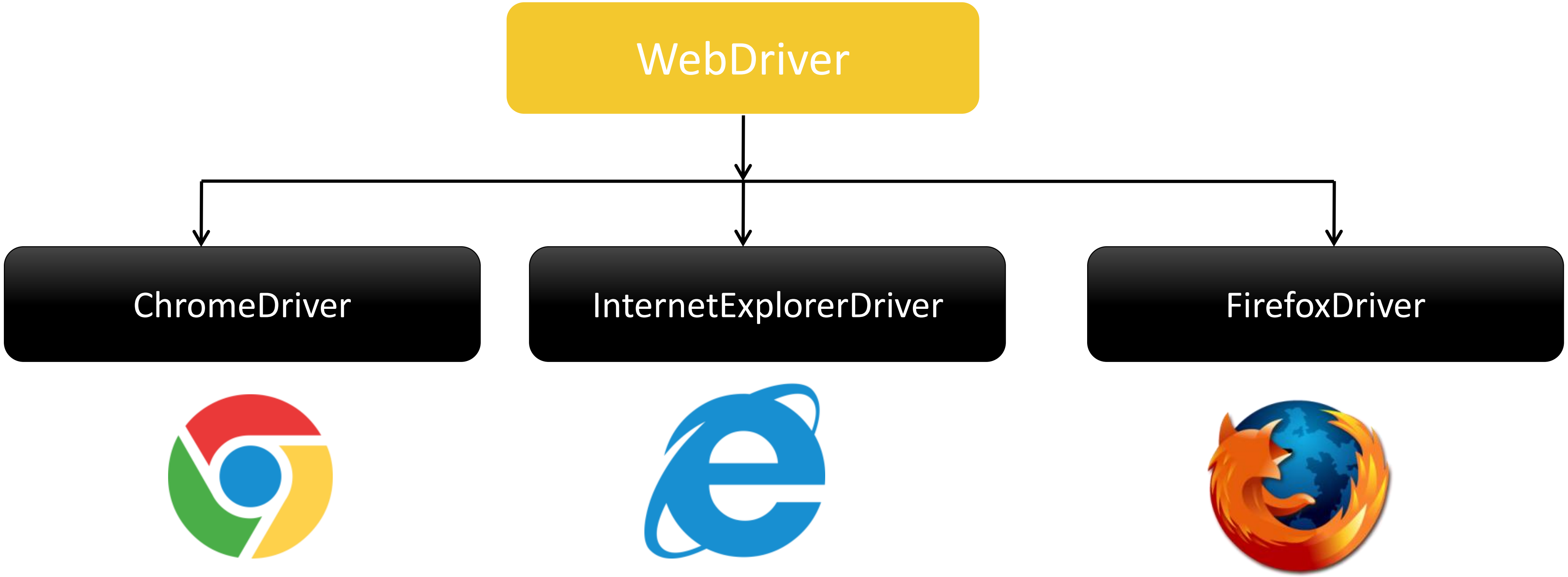
Hybrid inheritance in java by interface



class C **extends** A **implements** B1, B2
{
    //Implements methods from B1 & B2
}

```
class A1 {
void m1() {
System.out.println(" This is m1 from class A1");
}
}

interface B1 {
void m2();
}

interface B2 {
void m3();
}

class C extends A1 implements B1, B2 {
public void m2() {
System.out.println(" This is m2 from interface B1");
}

public void m3() {
System.out.println(" This is m3 from interface B2");
}
}

public class Test4 {
public static void main(String[] args) {

C cobj = new C();
cobj.m1();
cobj.m2();
cobj.m3();
}
}
```

Selenium WebDriver is an interface



WebDriver

ChromeDriver

InternetExplorerDriver

FirefoxDriver

Java Packages

- A **java package** is a group of similar types of classes, interfaces and sub-packages.
- Package in java can be categorized in two forms.
  - Built-in package
  - User-defined package
- There are many built-in packages such as java, lang, awt, javax, swing, net, io, util, sql etc.

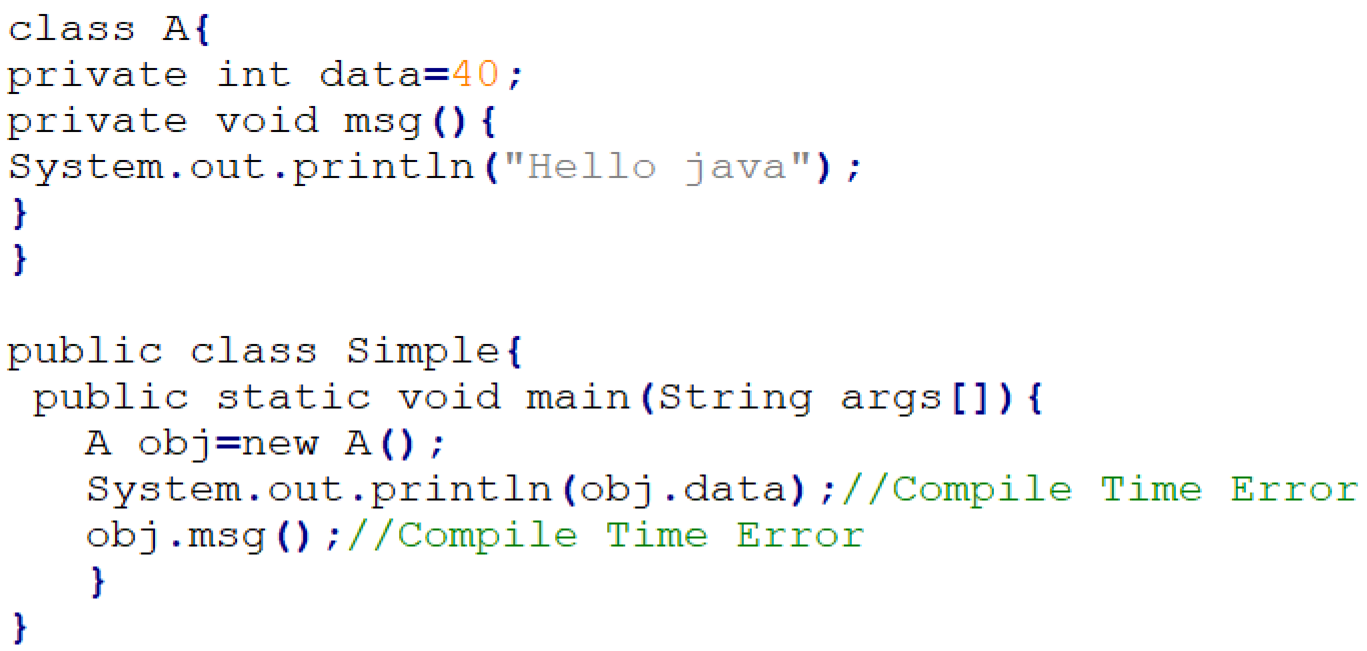
Access package from another package

- There are two ways to access the package from outside the package.
  - import package.*;
  - import package.classname;

Access Modifiers in java

- The access modifiers in java specifies accessibility (scope) of a data member, method, constructor or class.
- There are 4 types of java access modifiers:
  - private
  - default
  - protected
  - public

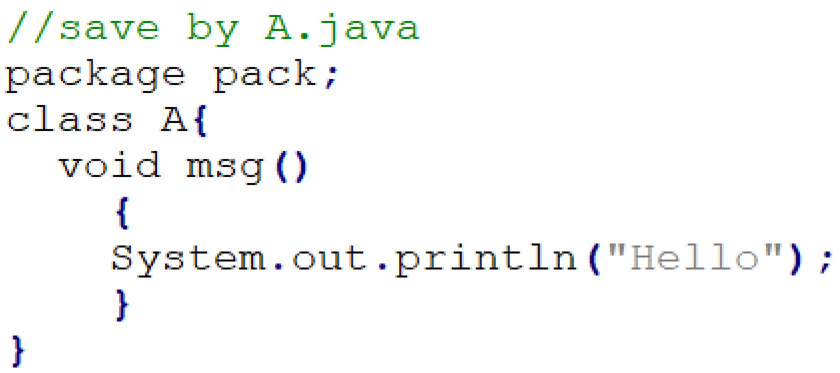
private access modifier



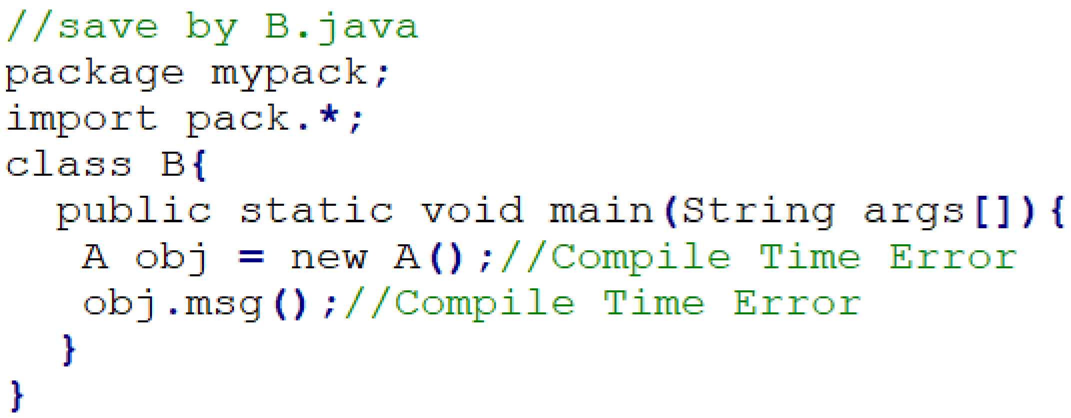
```
class A{
private int data=40;
private void msg(){
System.out.println("Hello java");
}
}

public class Simple{
 public static void main(String args[]){
   A obj=new A();
   System.out.println(obj.data);//Compile Time Error
   obj.msg();//Compile Time Error
   }
}
```

default access modifier

- If you don't use any modifier, it is treated as **default** by default. The default modifier is accessible only within package.



```
//save by A.java
package pack;
class A{
  void msg()
    {
    System.out.println("Hello");
    }
}
```

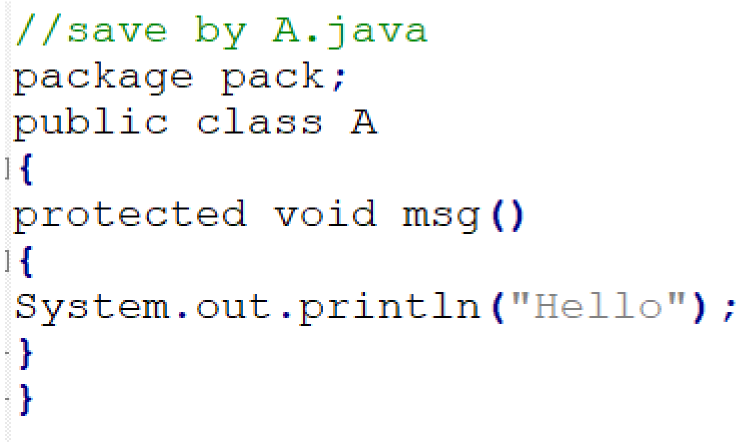


```
//save by B.java
package mypack;
import pack.*;
class B{
  public static void main(String args[]){
   A obj = new A();//Compile Time Error
   obj.msg();//Compile Time Error
  }
}
```

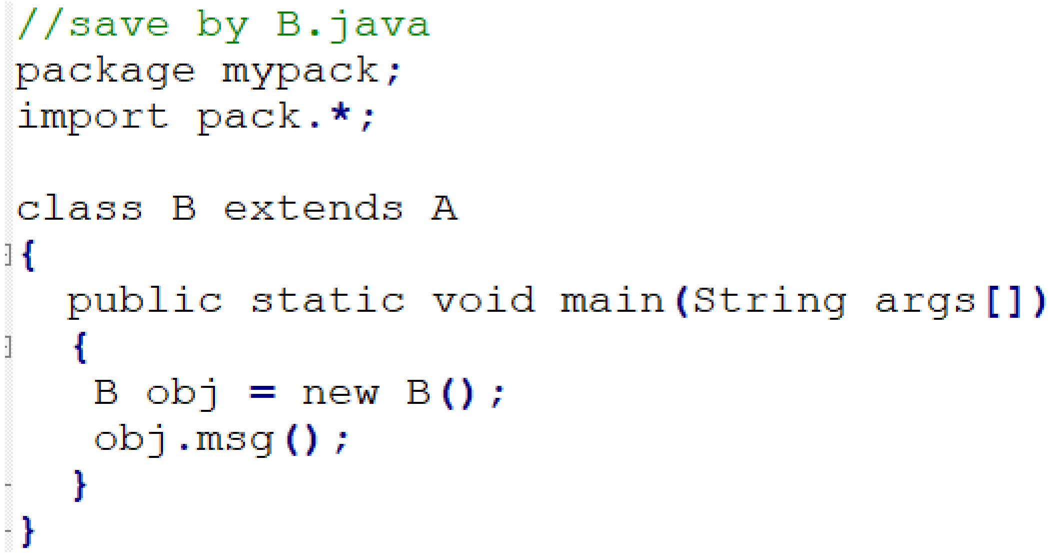
- * In the above example, the scope of class A and its method msg() is default so it cannot be accessed from outside the package.

protected access modifier

- The **protected access modifier** is accessible within package and outside the package but through inheritance only.
- The protected access modifier can be applied on the data member, method and constructor. It can't be applied on the class.



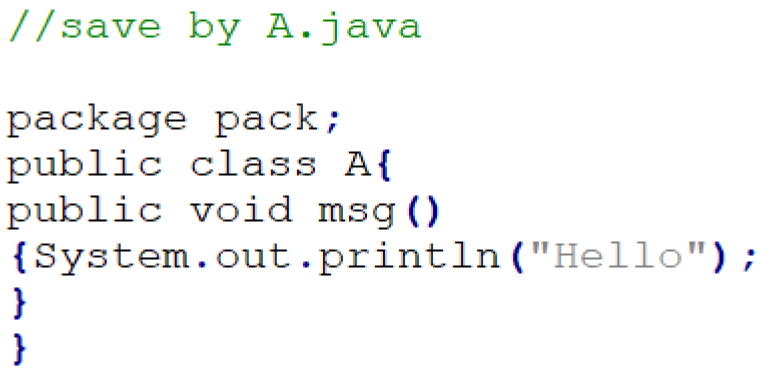
```
//save by A.java
package pack;
public class A
{
protected void msg()
{
System.out.println("Hello");
}
}
```



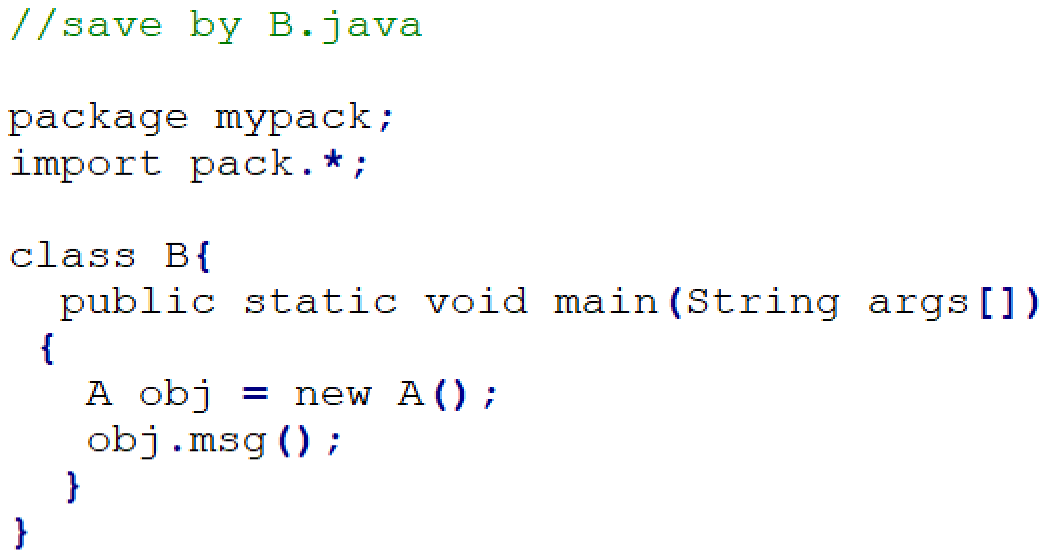
```
//save by B.java
package mypack;
import pack.*;

class B extends A
{
  public static void main(String args[])
  {
   B obj = new B();
   obj.msg();
  }
}
```

public access modifier

- The public access modifier is accessible everywhere. It has the widest scope among all other modifiers.



```java
//save by A.java

package pack;
public class A{
public void msg()
{System.out.println("Hello");
}
}
```



```java
//save by B.java

package mypack;
import pack.*;

class B{
  public static void main(String args[])
 {
   A obj = new A();
   obj.msg();
  }
}
```

Access modifiers

Access Modifier	within class	within package	outside package by subclass only	outside package
Private	**Y**	**N**	**N**	**N**
Default	**Y**	**Y**	**N**	**N**
Protected	**Y**	**Y**	**Y**	**N**
Public	**Y**	**Y**	**Y**	**Y**

Assignment

1. Write a program to demonstrate interface.
   - Interface A : int a, int b sum()
   - Class B : Implements method from A and calculate sum of a and b

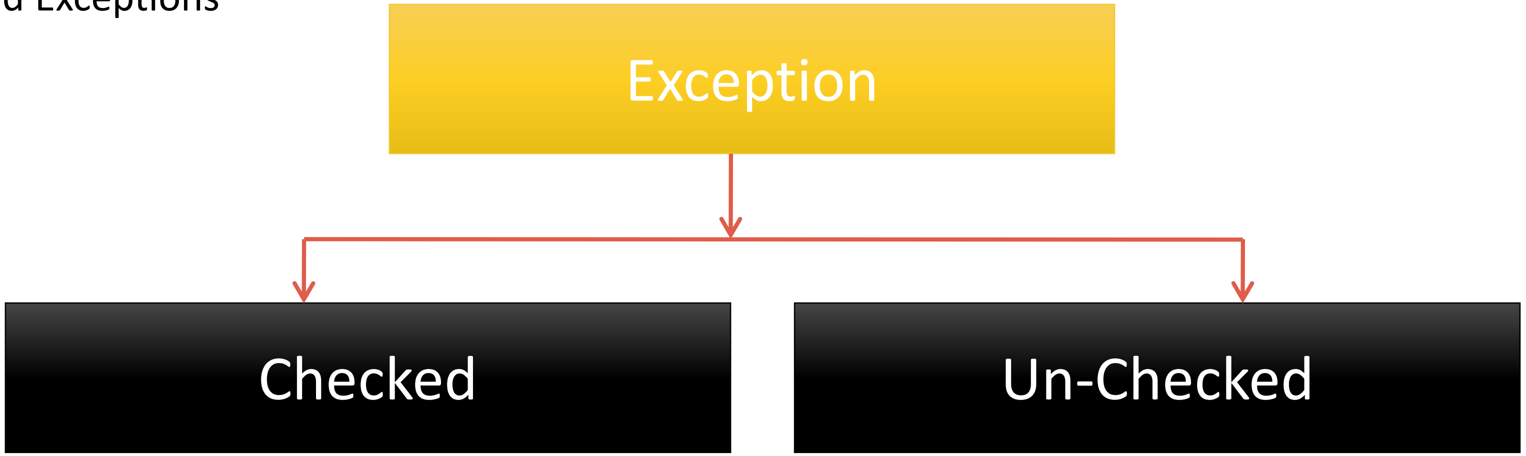
- 2. Write a program for multiple inheritance by using interface.
  - Interface A : int a, int b add()
  - Interface B : int x, int y mul()
  - Class Calculation : Implements methods from A and B interfaces.

Agenda

- Exception Handling

Java Exceptions

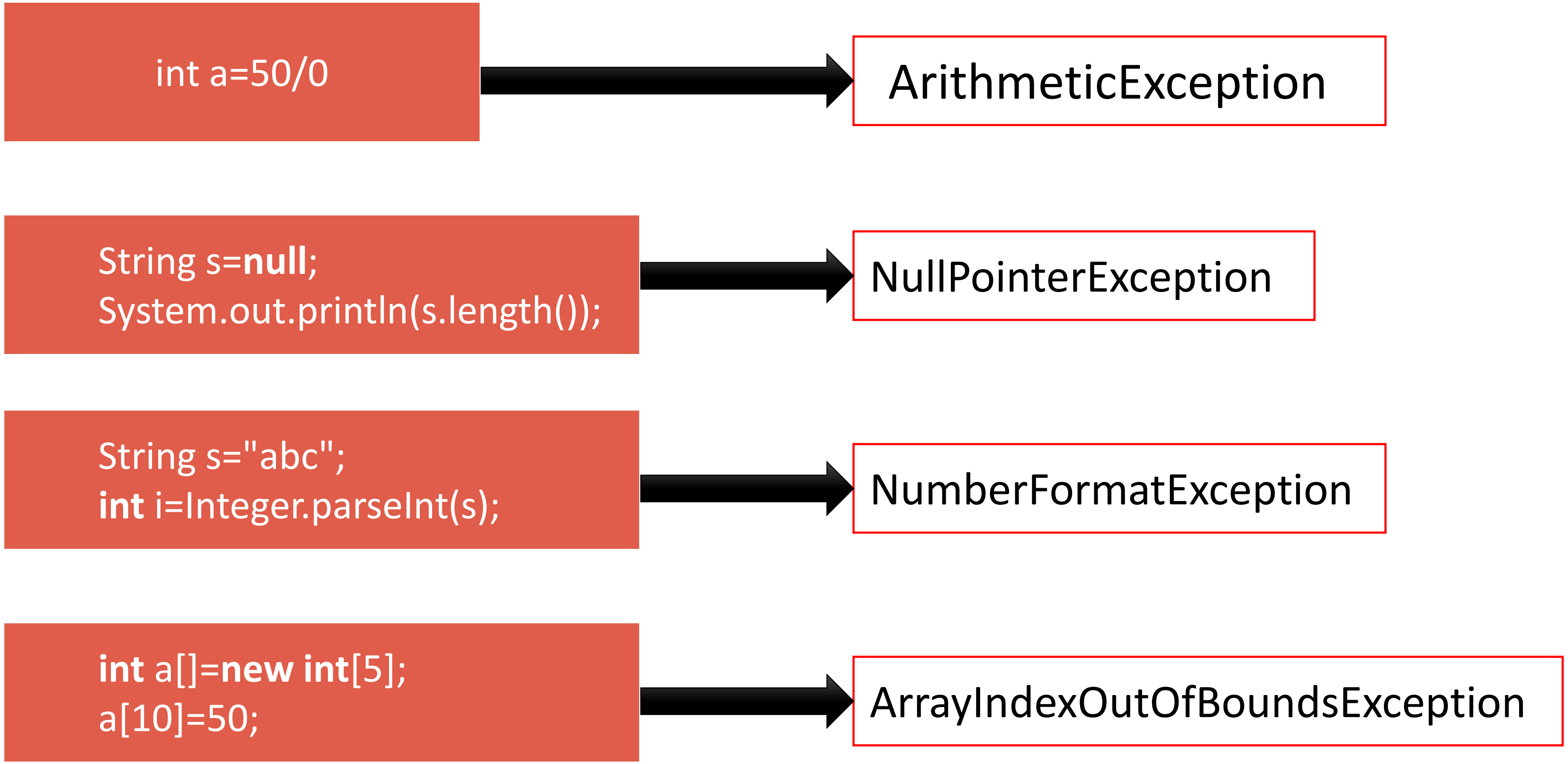
- Exception is an abnormal condition.
- In java, exception is an event that disrupts the normal flow of the program.
- There are two types of exceptions.
  1. Checked Exceptions
  2. Un-checked Exceptions



Un Checked Exceptions

- Exceptions that are NOT checked by compiler are called Un-Checked Exceptions.
- Un checked Exceptions successfully compiled by Java compiler.
- At run time it throws exception.
- Examples:
  - ArithmeticException
  - NullPointerException
  - NumberFormatException
  - ArrayIndexOutOfBoundsException

Common Un-Checked exceptions



```
int a=50/0
```

→ ArithmeticException

```
String s=null;
System.out.println(s.length());
```

→ NullPointerException

```
String s="abc";
int i=Integer.parseInt(s);
```

→ NumberFormatException

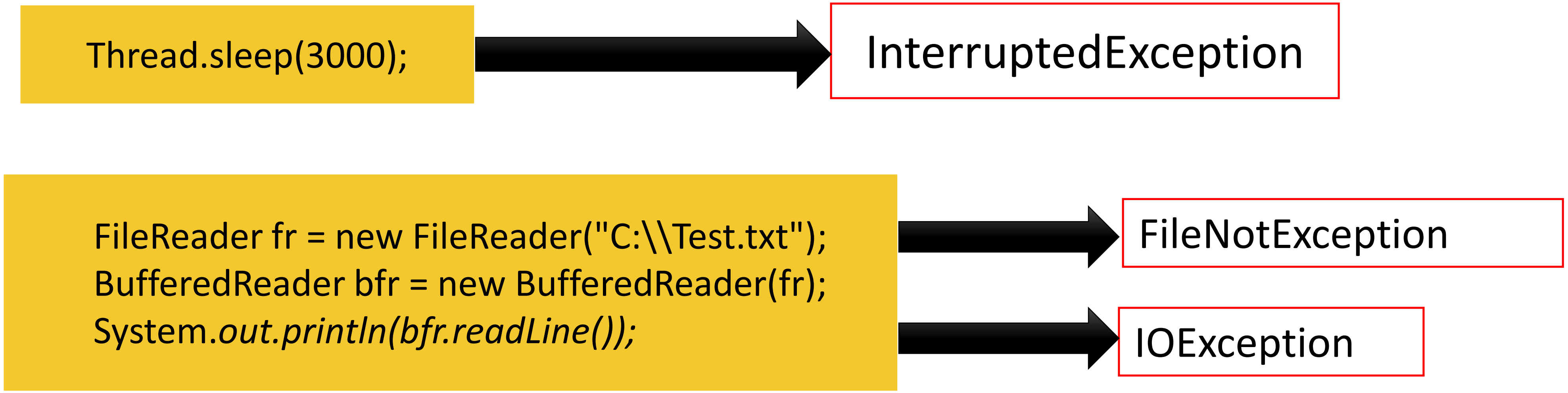
```
int a[]=new int[5];
a[10]=50;
```

→ ArrayIndexOutOfBoundsException

Checked Exceptions

- Exceptions that are checked by compiler are called Checked Exceptions.
- If a program contains checked-Exception code is not compiled.
- Examples:
  - InterruptedException
  - IOException
  - FileNotFoundException etc.

Common Checked exceptions



```
Thread.sleep(3000);
```

InterruptedException

```
FileReader fr = new FileReader("C:\\Test.txt");
BufferedReader bfr = new BufferedReader(fr);
System.out.println(bfr.readLine());
```

FileNotException

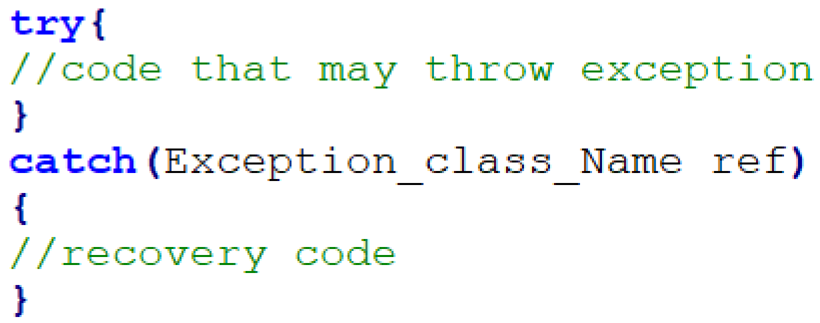
IOException

Java Exception Handling Keywords

- try
- catch
- finally
- throws

Java try..catch block

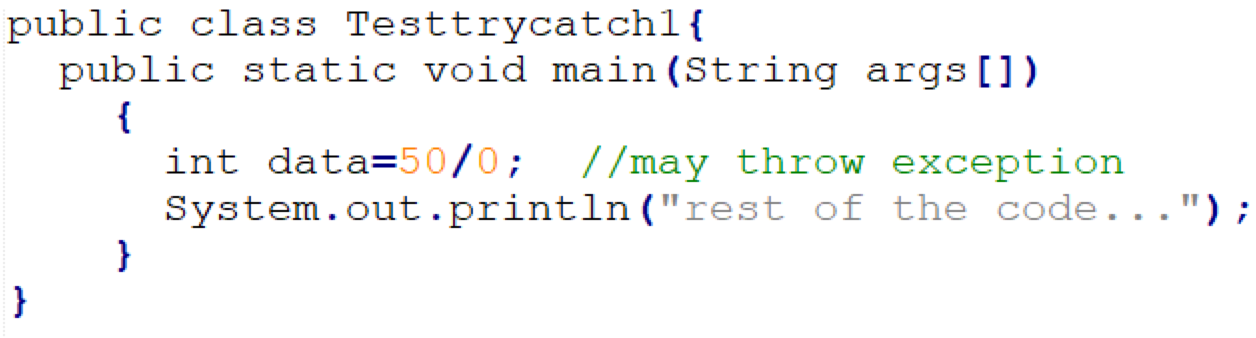
- Java try block is used to enclose the code that might throw an exception.
- It must be applied **at statement level within the method**.
- Java try block must be followed by either catch or finally block.
- Used for both **Un-checked and Checked Exceptions**.
- Java catch block is used to handle the Exception. It must be used after the try block only.
- You can use multiple catch block with a single try.



```
try{
//code that may throw exception
}
catch(Exception_class_Name ref)
{
//recovery code
}
```

Problem without exception handling

- Output: Exception in thread main java.lang.ArithmeticException:/ by zero

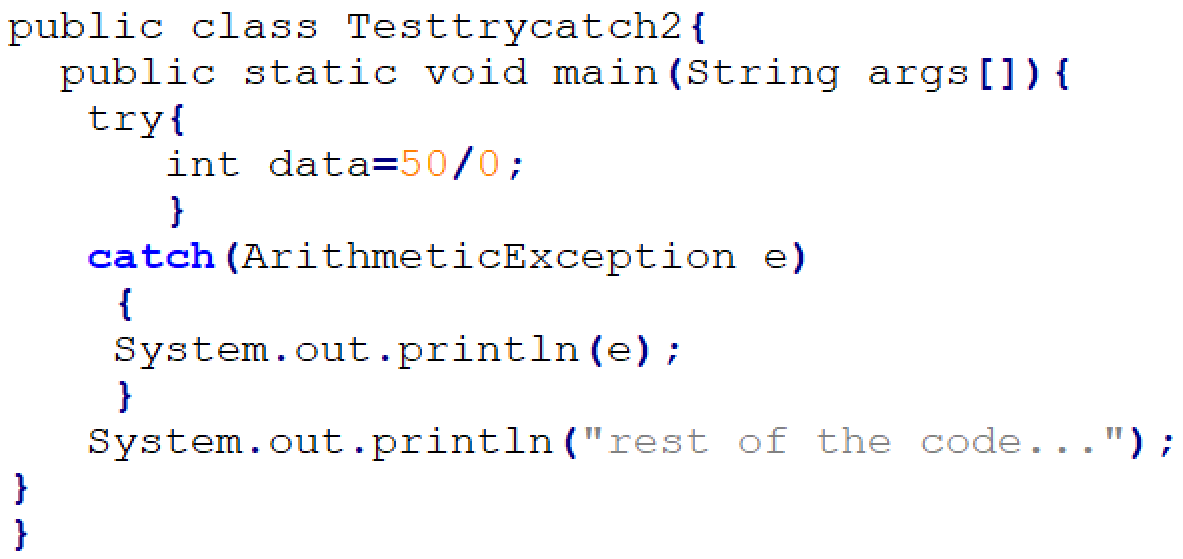


```
public class Testtrycatch1{
  public static void main(String args[])
    {
      int data=50/0;  //may throw exception
      System.out.println("rest of the code...");
    }
}
```

Solution by exception handling

Output: Exception in thread main java.lang.ArithmeticException:/ by zero

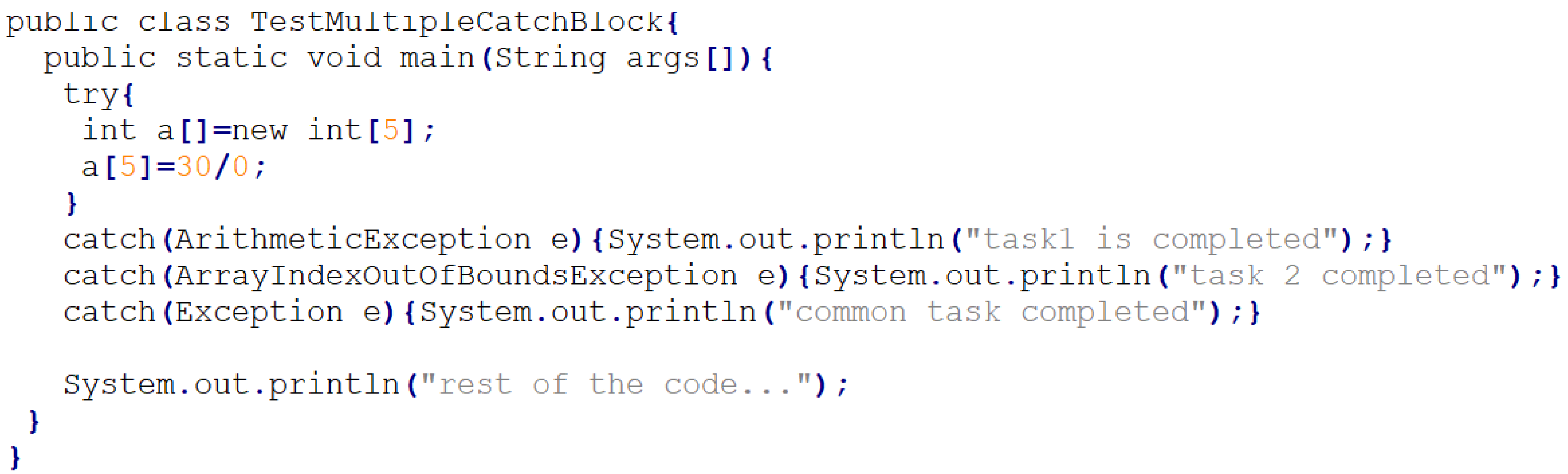
rest of the code...



```
public class Testtrycatch2{
  public static void main(String args[]){
   try{
      int data=50/0;
      }
   catch(ArithmeticException e)
    {
    System.out.println(e);
    }
   System.out.println("rest of the code...");
}
}
```

Java Multi catch block

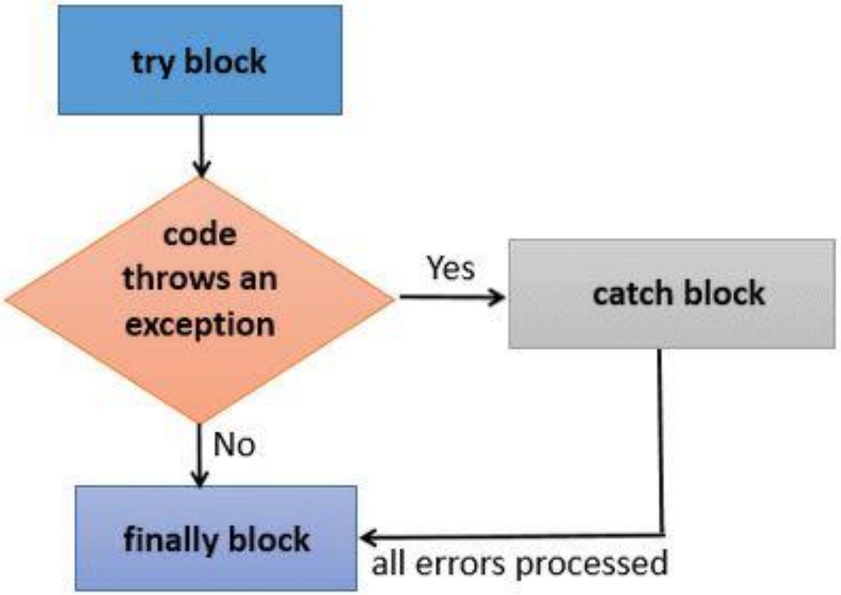
- If you have to perform different tasks at the occurrence of different Exceptions, use java multi catch block.



```
public class TestMultipleCatchBlock{
  public static void main(String args[]){
   try{
    int a[]=new int[5];
    a[5]=30/0;
   }
   catch(ArithmeticException e){System.out.println("task1 is completed");}
   catch(ArrayIndexOutOfBoundsException e){System.out.println("task 2 completed");}
   catch(Exception e){System.out.println("common task completed");}

   System.out.println("rest of the code...");
 }
}
```

Java finally block

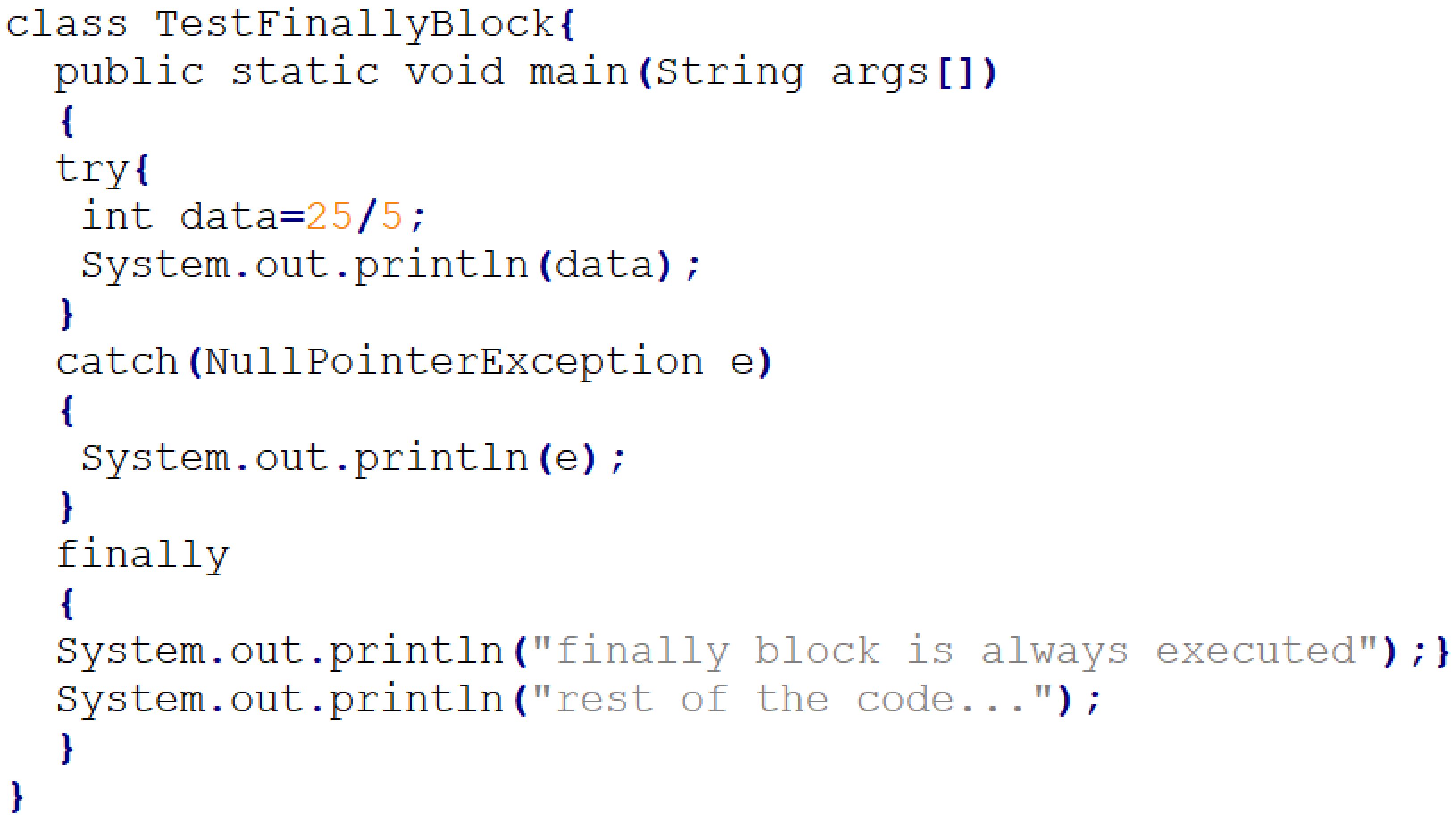
- **Java finally block** is a block that is used *to execute important code* such as closing connection, stream etc.
- Java finally block is always executed whether exception is handled or not.
- Java finally block follows try or catch block.



Usage of Java finally

- Cases
  1. Exception doesn't occur.
  2. Exception occurs and not handled.
  3. Exception occurs and handled.

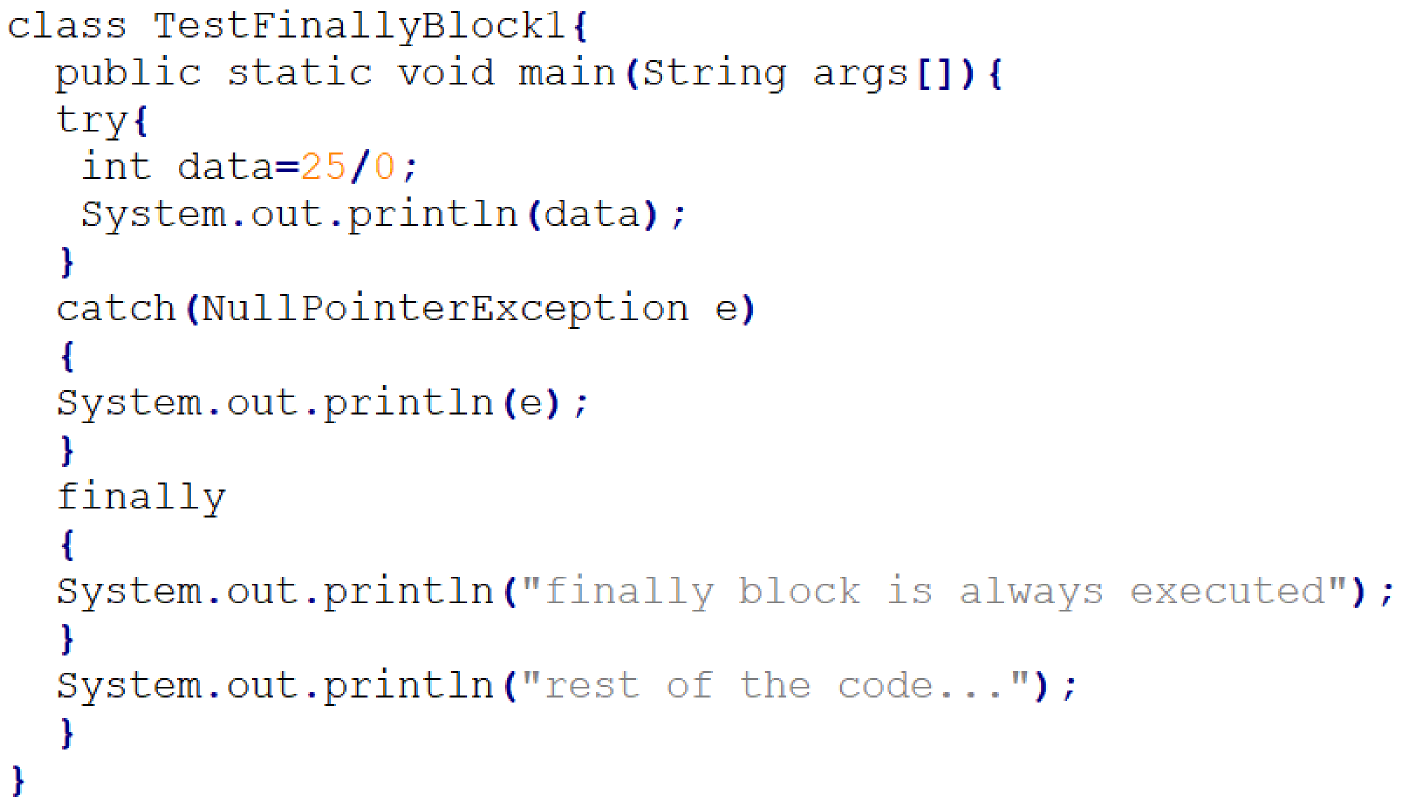
Case 1: Java finally example where exception doesn't occur



```
class TestFinallyBlock{
  public static void main(String args[])
  {
  try{
   int data=25/5;
   System.out.println(data);
  }
  catch(NullPointerException e)
  {
   System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");}
  System.out.println("rest of the code...");
  }
}
```

Case 2: Java finally example where exception occurs and not handled.

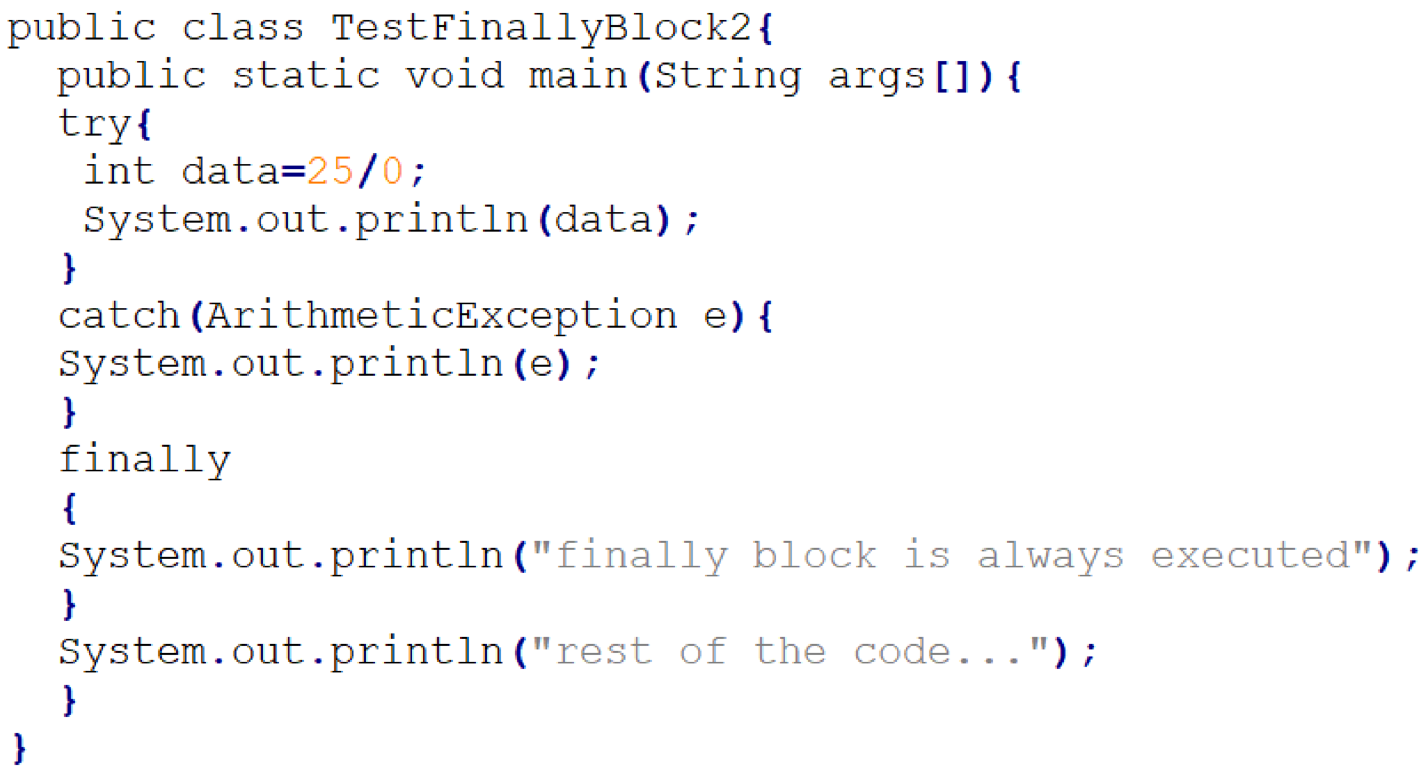
- Output:finally block is always executed
- Exception in thread main java.lang.ArithmeticException:/ by zero



```
class TestFinallyBlock1{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(NullPointerException e)
  {
  System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");
  }
  System.out.println("rest of the code...");
  }
}
```

Case 3: Java finally example where exception occurs and handled.

Output: Exception in thread main java.lang.ArithmeticException:/ by zero finally block is always executed   rest of the code...

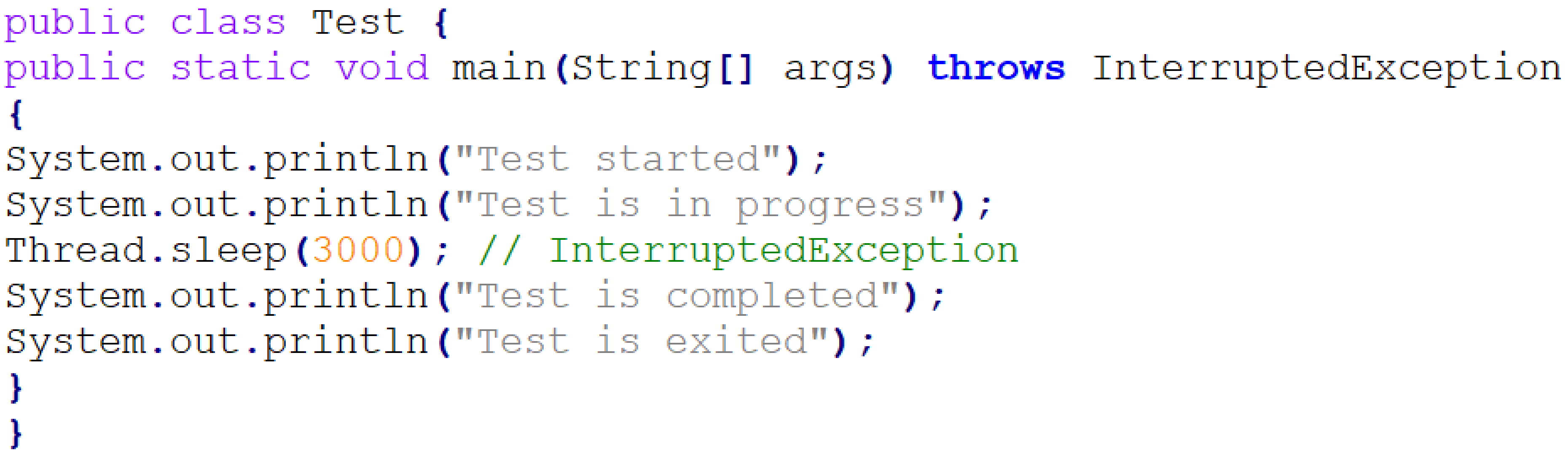


```
public class TestFinallyBlock2{
  public static void main(String args[]){
  try{
   int data=25/0;
   System.out.println(data);
  }
  catch(ArithmeticException e){
  System.out.println(e);
  }
  finally
  {
  System.out.println("finally block is always executed");
  }
  System.out.println("rest of the code...");
  }
}
```

throws

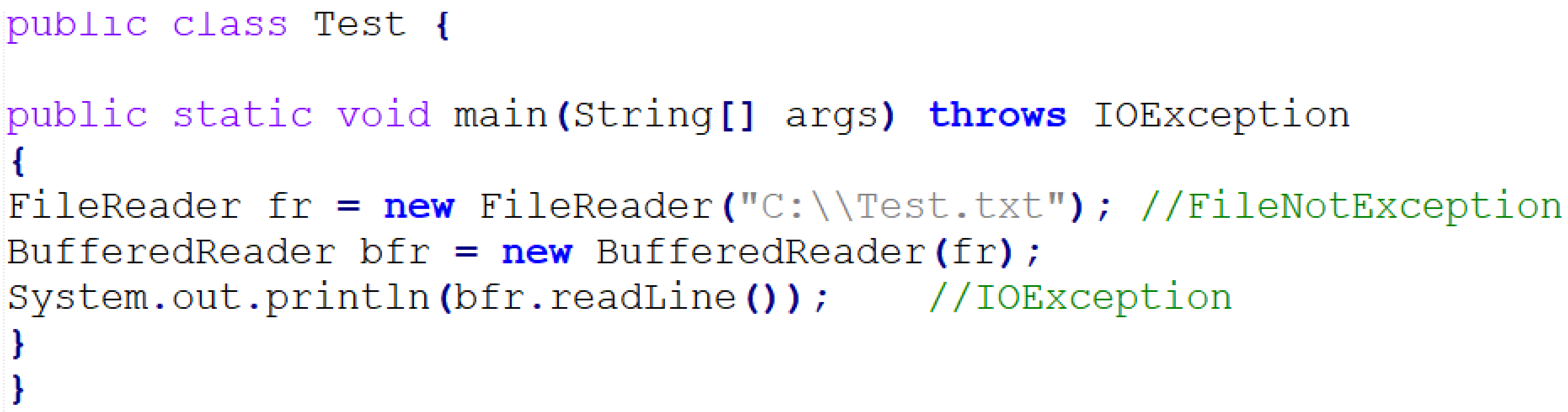
- Used for only Checked Exceptions.
- It should be applied at Method level.

throws – Example1



```
public class Test {
public static void main(String[] args) throws InterruptedException
{
System.out.println("Test started");
System.out.println("Test is in progress");
Thread.sleep(3000); // InterruptedException
System.out.println("Test is completed");
System.out.println("Test is exited");
}
}
```

throws – Example2



```
public class Test {

public static void main(String[] args) throws IOException
{
FileReader fr = new FileReader("C:\\Test.txt"); //FileNotException
BufferedReader bfr = new BufferedReader(fr);
System.out.println(bfr.readLine());    //IOException
}
}
```

	Un-Checked	Checked	Method Level	Within the method
Try..Catch	Y	Y	N	Y
throws	N	Y	Y	N

Assingment

1. Write a java program for the following and handle exceptions by using try..catch and finally blocks.
   - Any number divide by zero.
   - int a[]=null;
   - a.length
   - String s="abc";
   - int i=Integer.parseInt(s);

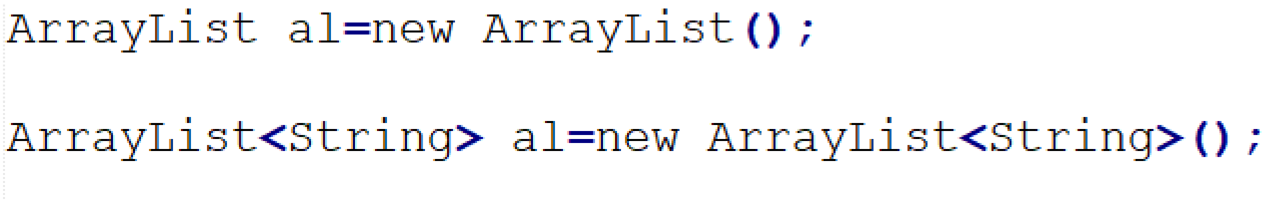
2. Write a java program to handle IO Exception by using *throws*.

Agenda

- ArrayList
- HashMap
- JDBC

ArrayList

- ArrayList is pre defined class in Java used for dynamic array for storing elements.
- ArrayList can contains duplicate elements.
- We can add, insert and remove elements from ArrayList.

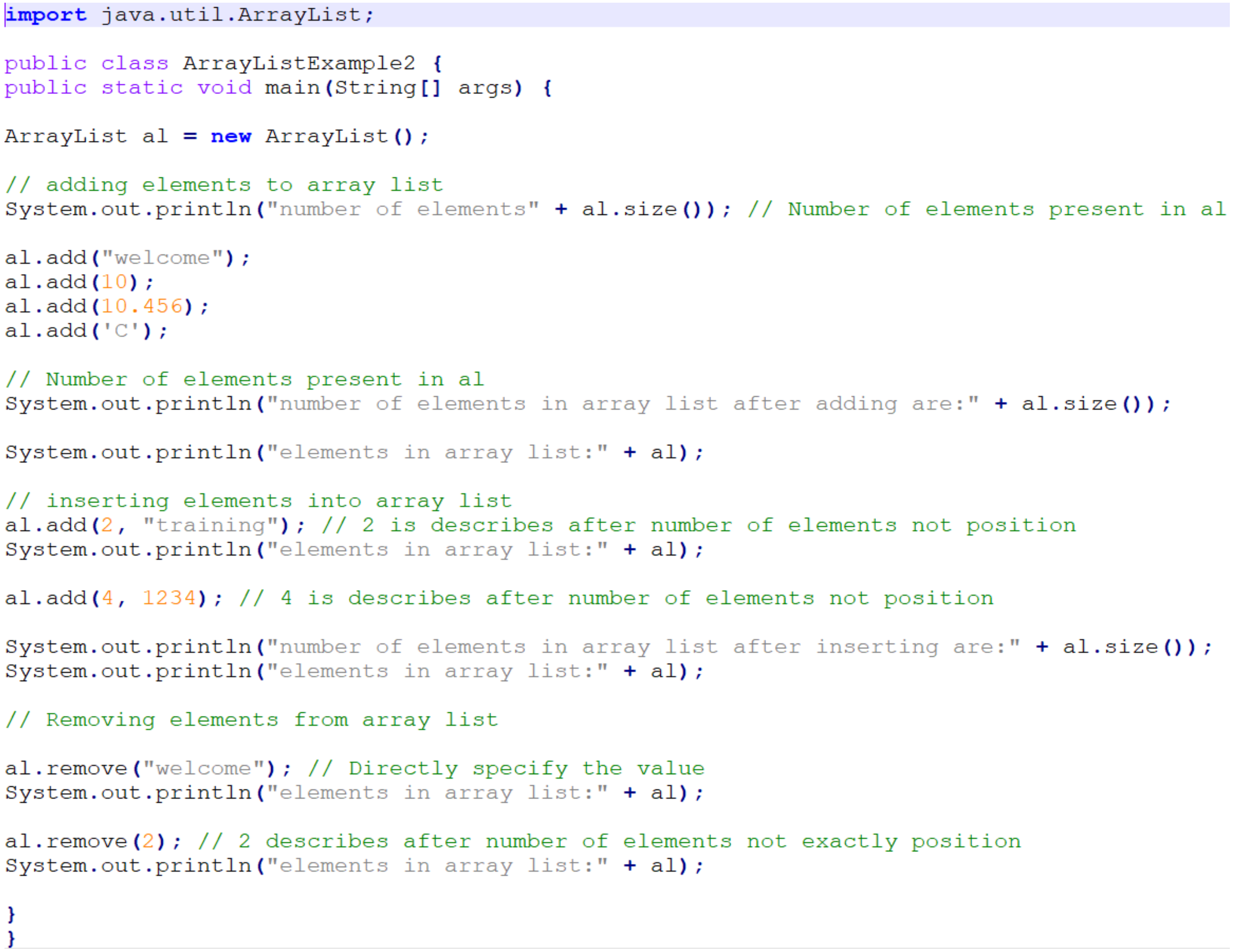


```
ArrayList al=new ArrayList();

ArrayList<String> al=new ArrayList<String>();
```

Java ArrayList Example1

```java
import java.util.ArrayList;
public class ArrayListExample {

public static void main(String[] args) {

ArrayList<String> list = new ArrayList<String>();

// adding elements to array list
list.add("Raj");
list.add("Ravi");
list.add("Pavan");
list.add("Simran");
list.add("Arvinder");

System.out.println(list.size());    // returns number of elements in array list

for (String s : list) // reading elements from array list
{
System.out.println(s);
}
}
}
```

Java ArrayList Example2

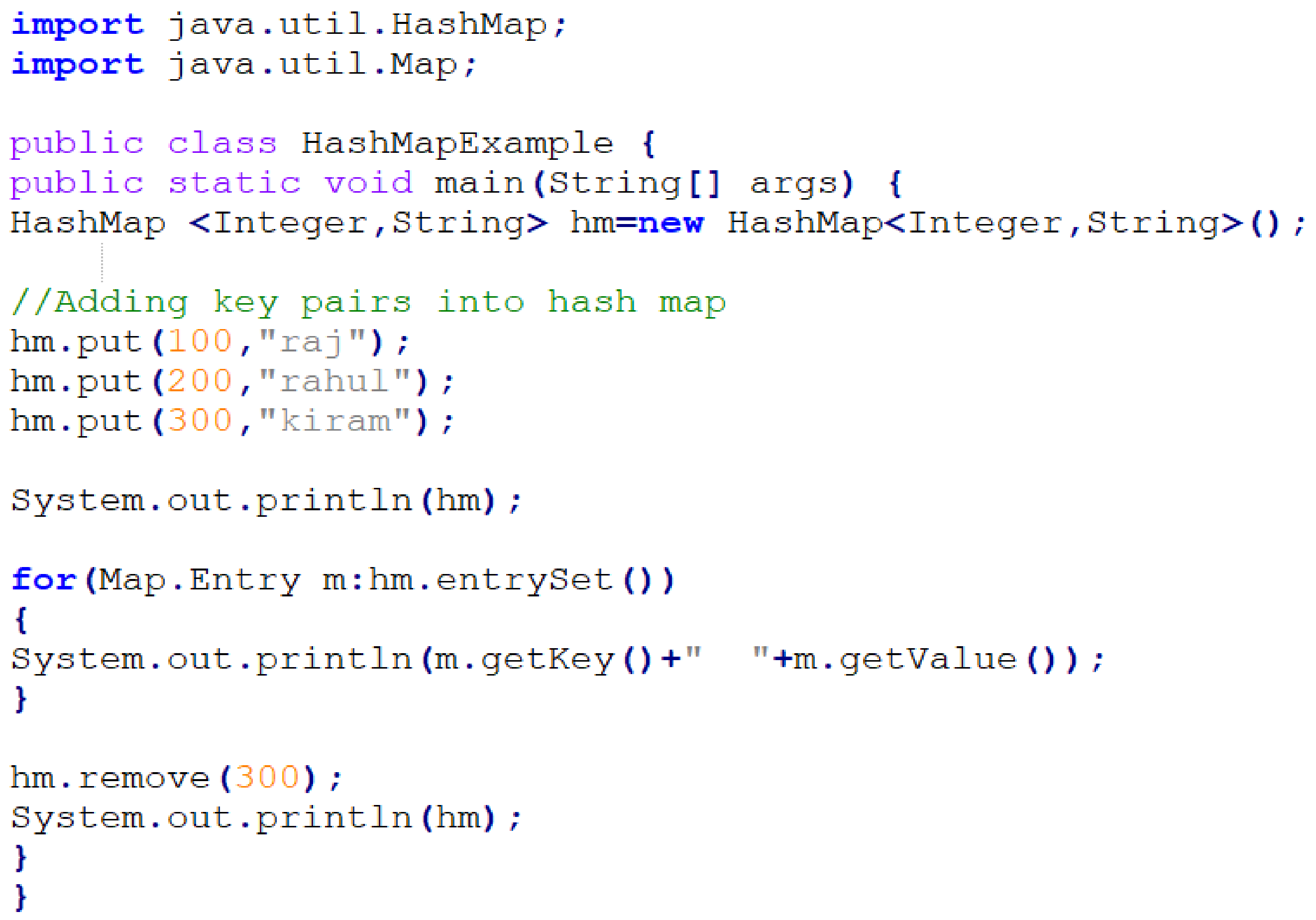


```
import java.util.ArrayList;

public class ArrayListExample2 {
public static void main(String[] args) {

ArrayList al = new ArrayList();

// adding elements to array list
System.out.println("number of elements" + al.size()); // Number of elements present in al

al.add("welcome");
al.add(10);
al.add(10.456);
al.add('C');

// Number of elements present in al
System.out.println("number of elements in array list after adding are:" + al.size());

System.out.println("elements in array list:" + al);

// inserting elements into array list
al.add(2, "training"); // 2 is describes after number of elements not position
System.out.println("elements in array list:" + al);

al.add(4, 1234); // 4 is describes after number of elements not position

System.out.println("number of elements in array list after inserting are:" + al.size());
System.out.println("elements in array list:" + al);

// Removing elements from array list

al.remove("welcome"); // Directly specify the value
System.out.println("elements in array list:" + al);

al.remove(2); // 2 describes after number of elements not exactly position
System.out.println("elements in array list:" + al);

}
}
```

HashMap

- The important points about Java HashMap:
  - A HashMap contains values based on the key.
  - It contains only unique elements.
  - It maintains no order.

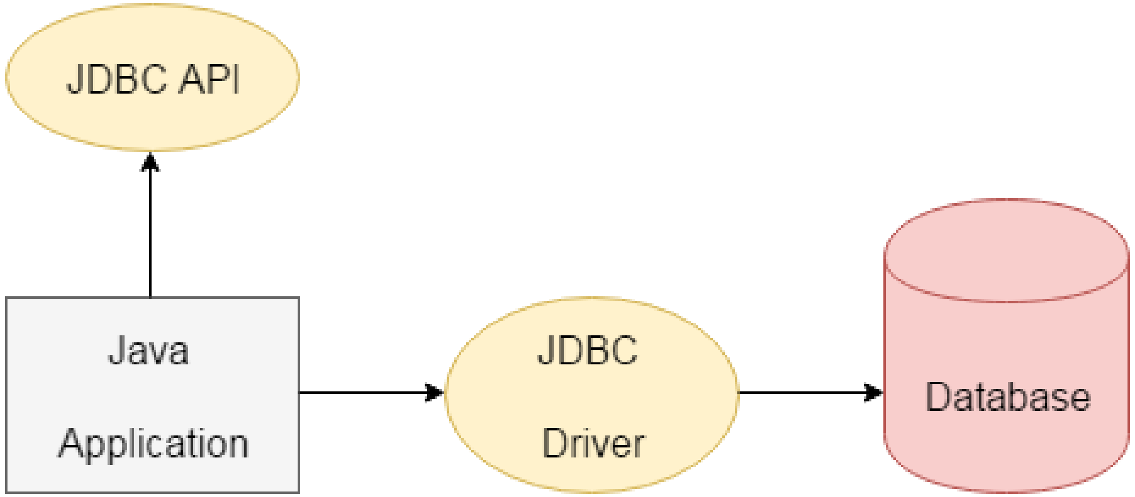
Java HashMap Example



```
import java.util.HashMap;
import java.util.Map;

public class HashMapExample {
public static void main(String[] args) {
HashMap <Integer,String> hm=new HashMap<Integer,String>();

//Adding key pairs into hash map
hm.put(100,"raj");
hm.put(200,"rahul");
hm.put(300,"kiram");

System.out.println(hm);

for(Map.Entry m:hm.entrySet())
{
System.out.println(m.getKey()+"  "+m.getValue());
}

hm.remove(300);
System.out.println(hm);
}
}
```

JDBC – Java Database Connectivity

- Java JDBC is a java API to connect and execute query with the database.
- JDBC API uses jdbc drivers to connect with the database.



Database and SQL

- Database: stores the data in the tables.
- SQL- a language used for communicate to the database.
  - DML : Data Manipulation Language
  - DDL : Data Definition Language
  - DCL : Data Control Language
  - TCL : Transaction Language

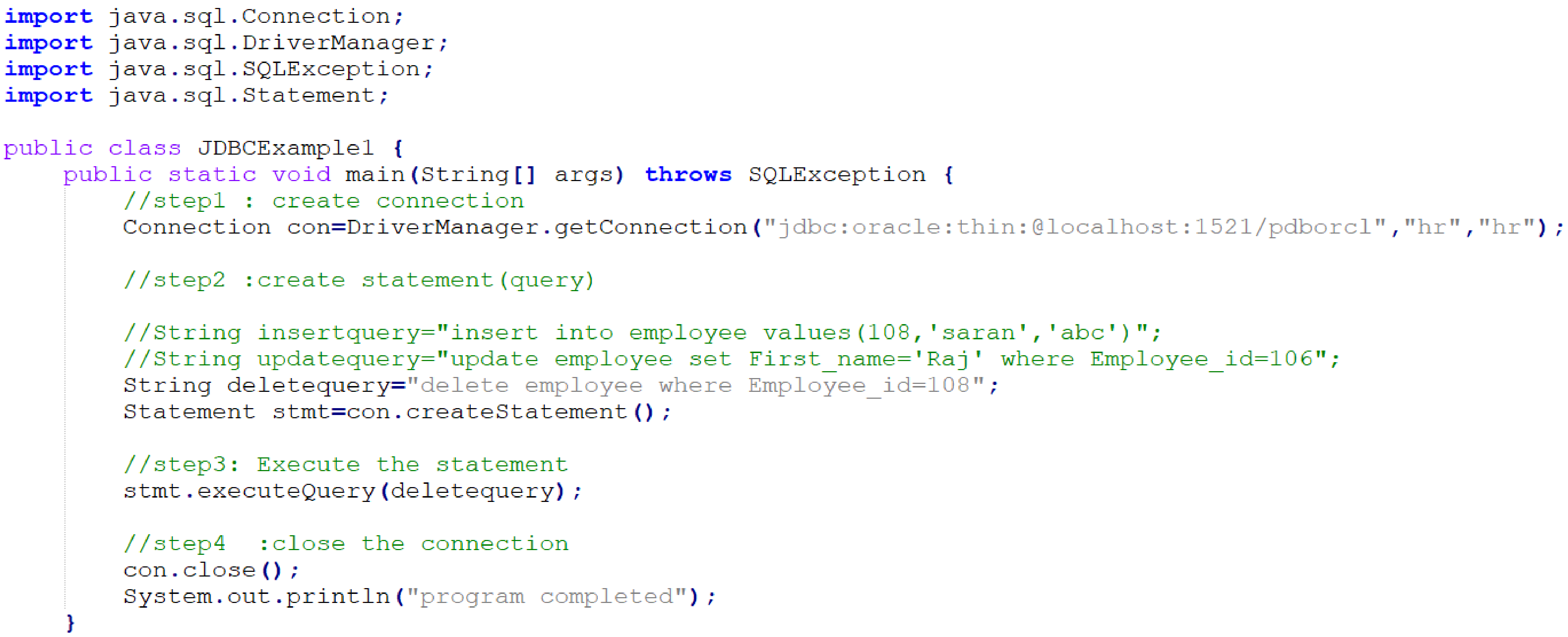
Database Components

- Database Client
  - CLI
  - GUI
- Database Server

4 Steps to connect to the database in java

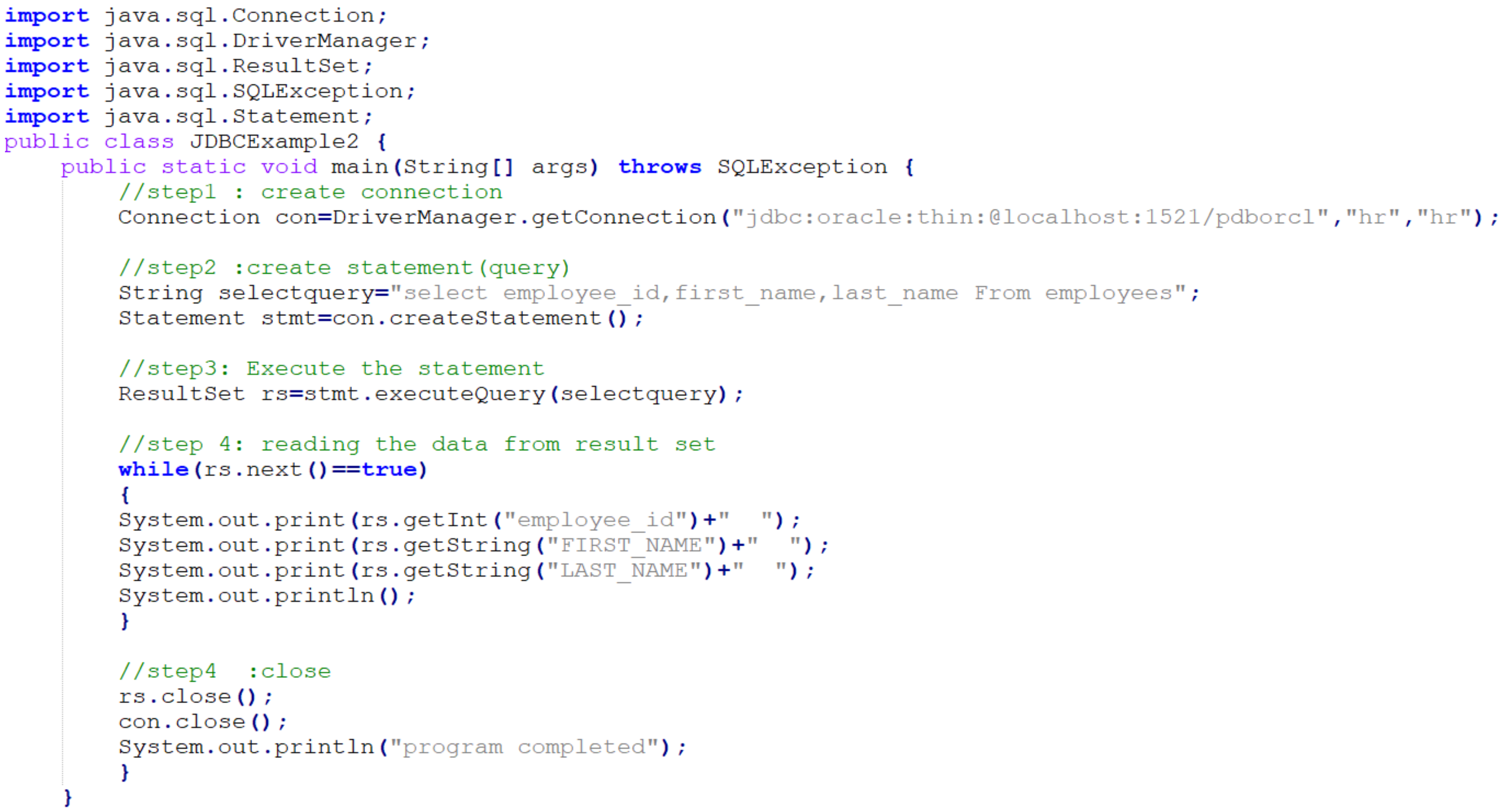
- Creating connection
- Creating statement
- Executing queries
- Closing connection

JDBC Example1



```java
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
import java.sql.Statement;

public class JDBCExample1 {
    public static void main(String[] args) throws SQLException {
        //step1 : create connection
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/pdborcl","hr","hr");

        //step2 :create statement(query)

        //String insertquery="insert into employee values(108,'saran','abc')";
        //String updatequery="update employee set First_name='Raj' where Employee_id=106";
        String deletequery="delete employee where Employee_id=108";
        Statement stmt=con.createStatement();

        //step3: Execute the statement
        stmt.executeQuery(deletequery);

        //step4  :close the connection
        con.close();
        System.out.println("program completed");
    }
```

JDBC Example2



```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Statement;
public class JDBCExample2 {
    public static void main(String[] args) throws SQLException {
        //step1 : create connection
        Connection con=DriverManager.getConnection("jdbc:oracle:thin:@localhost:1521/pdborcl","hr","hr");

        //step2 :create statement(query)
        String selectquery="select employee_id,first_name,last_name From employees";
        Statement stmt=con.createStatement();

        //step3: Execute the statement
        ResultSet rs=stmt.executeQuery(selectquery);

        //step 4: reading the data from result set
        while(rs.next()==true)
        {
        System.out.print(rs.getInt("employee_id")+"   ");
        System.out.print(rs.getString("FIRST_NAME")+"   ");
        System.out.print(rs.getString("LAST_NAME")+"   ");
        System.out.println();
        }

        //step4  :close
        rs.close();
        con.close();
        System.out.println("program completed");
        }
    }
```