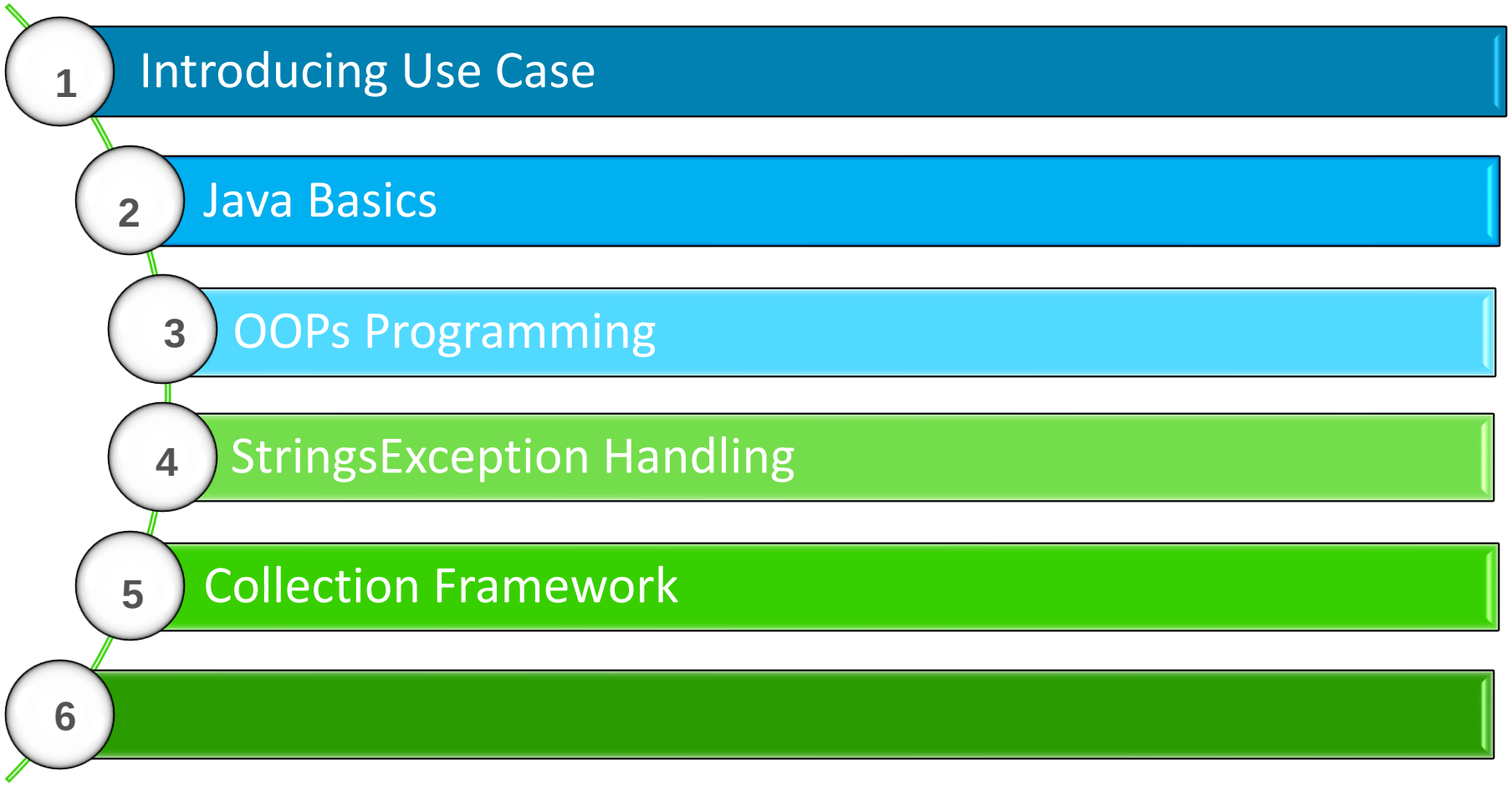
axess academy

Java Programming

axess academy

Modules



axess academy

Introducing the Use Case

axess academy

Bank Customer Management System

- Display the menu to user

- Create the customer class having fields cutsomerid as int, name as string, mailid as string contact as string and account type as string.

- If user enters 1 , please provide the details. On valid entries these details should be added into the list with auto generated customerid.

```
Welcome to Standard Chartered Bank
Please enter your choice
1 for Add new Customer
2 for Display Customers
3 for Search Customer
4 for Delete Customer
5 for Exit the bank application
```

```
Please enter customer details :
Enter name :
Sandra
Enter email :
Sandra@gmail.com
Enter contact :
9988778877
Enter account type (Savings or Current):
Savings
Customer added suucessfully with customer id 4656
```

axess academy

Bank Customer Management System …

- While adding the customer, make sure admin provides valid entries like name can only have alphabets, email should be valid one , contact no should have 10 digits and account type can be either Savings or current. If any one of the condition fails, the customer should not be added and display the appropriate error message.
- If user enters 2 , display all existing customer details using collection.

```
Customer Id = 7262, Customer name = Sandra, Customer email = Sandra@gmail.com, Customer conta
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer c
Customer Id = 2680, Customer name = Steve, Customer email = Steve@gmail.com, Customer contact
```

- If user enters 3 , display the matched customer details.

```
3
Please enter customer id
1014
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer
```

- If user enters 4 , delete the matched customer details.

```
4
Please enter customer id to be deleted
1014
Deleted customer with id =1014
```

- If user enters 5, exit the application.

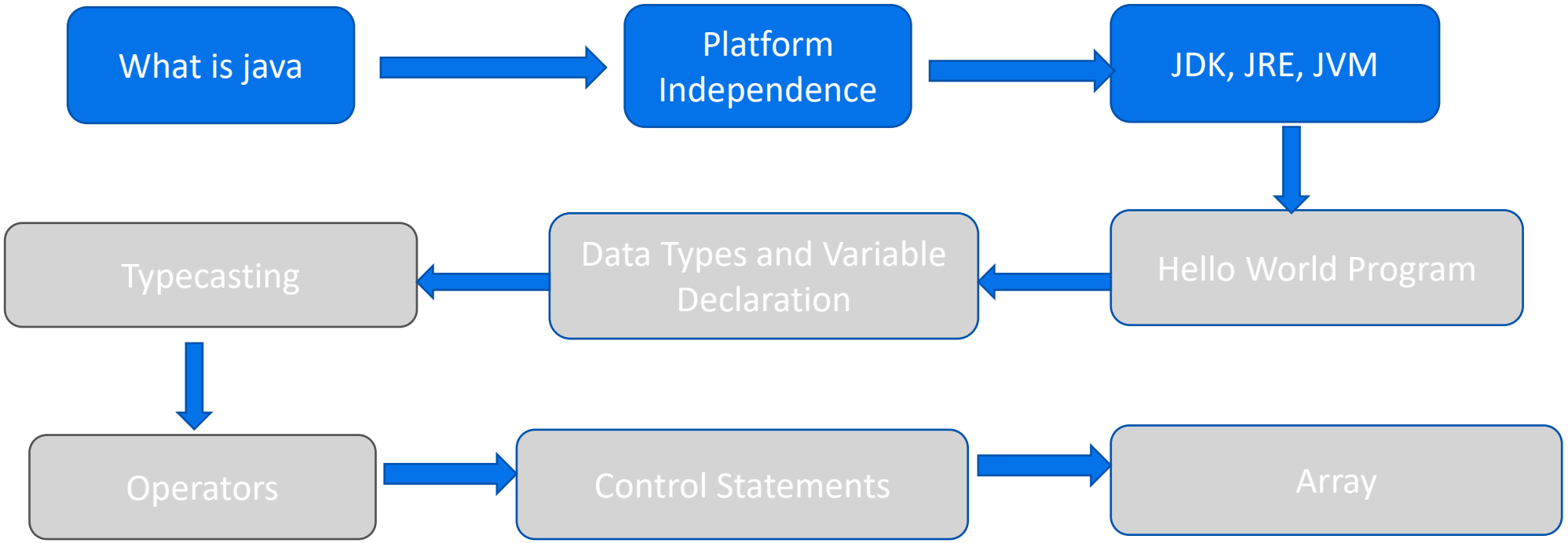
axess academy

Module 1

Java Basics

axess academy

Module 1 : Java Basics Design



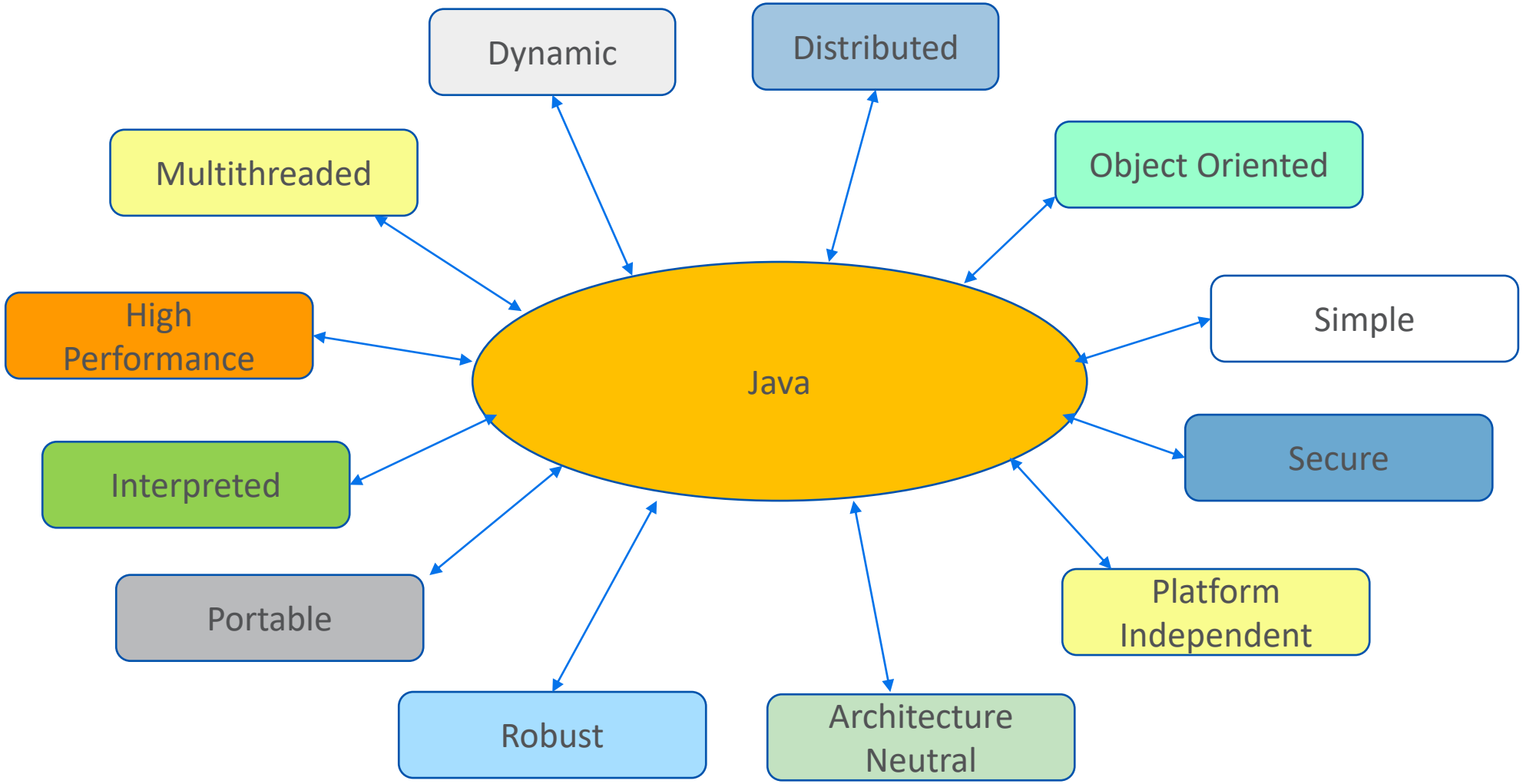
What is Java

Java

- Java is one of the most popular and widely used programming language and platform.
- A platform is an environment that helps to develop and run programs written in any programming language.
- **Java** is a general-purpose programming language that is class-based, object-oriented (although not a pure OO language, as it contains primitive types)
- It is intended to let application developers *write once, run anywhere* (WORA), meaning that compiled Java code can run on all platforms that support Java without the need for recompilation.
- Java was originally developed by James Gosling at Sun Microsystems (which has since been acquired by Oracle) and released in 1995 as a core component of Sun Microsystems' Java platform.
- The latest version is Java 22, released in March 2024.

axess academy

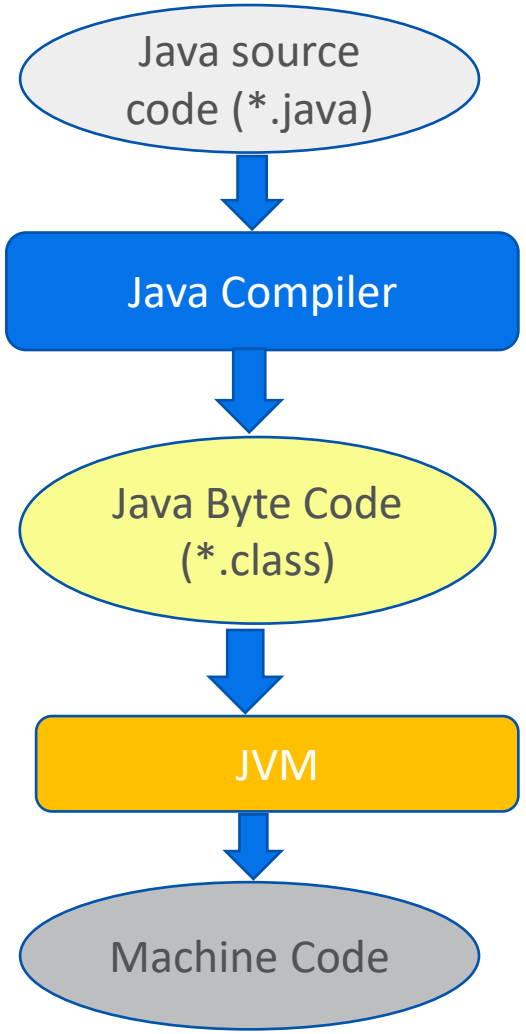
Java Features



axess academy

Java as Platform Independent

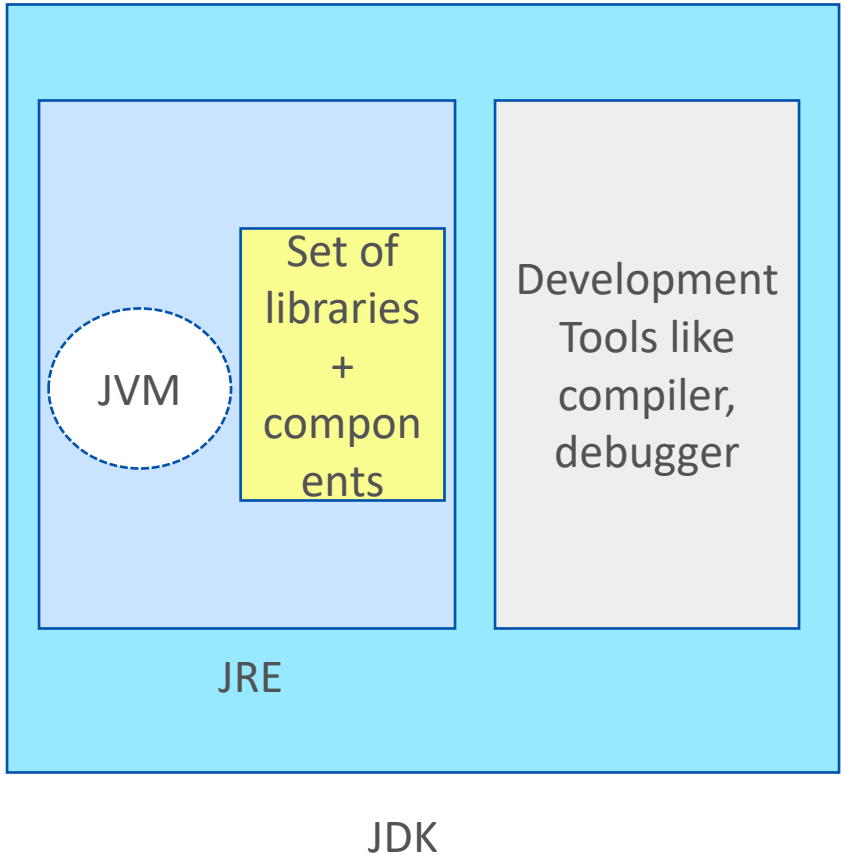
- **Platform independent** language means once compiled you can execute the program on any **platform** (**OS**).
- The Java compiler converts the source code to bytecode, which is an intermediate Language.
- The bytecode generated is a non-executable code and needs an interpreter to execute on a machine. This interpreter is the JVM and thus the Bytecode is executed by the JVM.
- **Java is platform independent but JVM is platform dependent.**



axess academy

JDK, JRE and JVM

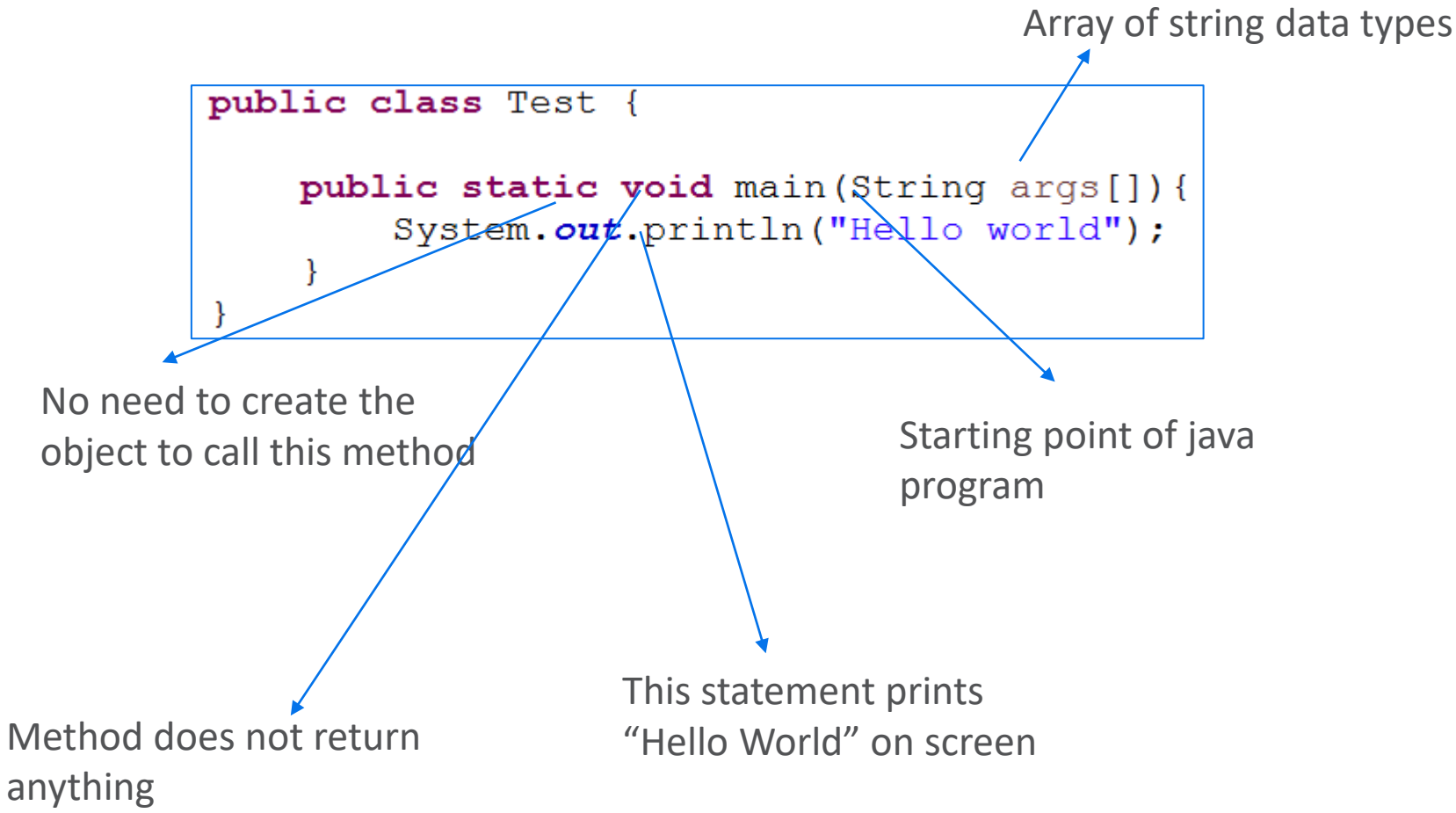
- JRE and JDK come as installer while JVM is bundled with them.
- The JDK also called Java Development Kit is a superset of the JRE, and contains everything that is in the JRE, plus tools such as the compilers and debuggers necessary for developing applications.
- The Java Runtime Environment (JRE) provides the libraries, the Java Virtual Machine, and other components to run applications. JRE does not contain tools and utilities such as compilers or debuggers for developing applets and applications.
- Java Virtual Machine(JVM) is platform dependent and it interprets the Byte Code into specific Machine language of the platform.



Hello World Program

axess academy

Hello World Program



axess academy

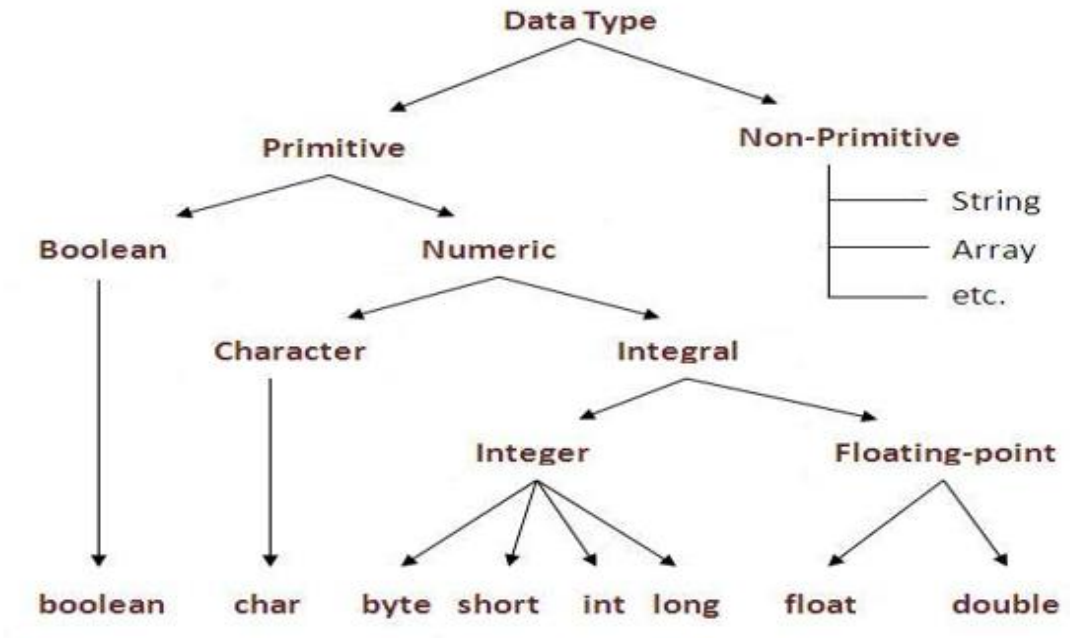
Data Types and variable declaration

axess academy

Data Types

- Java language has a rich implementation of data types.Data types specify size and the type of values that can be stored in an identifier.

- Data types are classified into two categories
  - Primitive Data type
  - Non-Primitive Data type



axess academy

Variable Declaration

- A variable is the name given to a memory location. It is the basic unit of storage in a program.
- The value stored in a variable can be changed during program execution.
- A variable is only a name given to a memory location so all the operations done on the variable effects that memory location.
- In Java, all the variables must be declared before use.
- Example:

```
int  age=20;
boolean  flag = false;
double amount= 2548.78;
String s= “Hello World”;
```

Typecasting

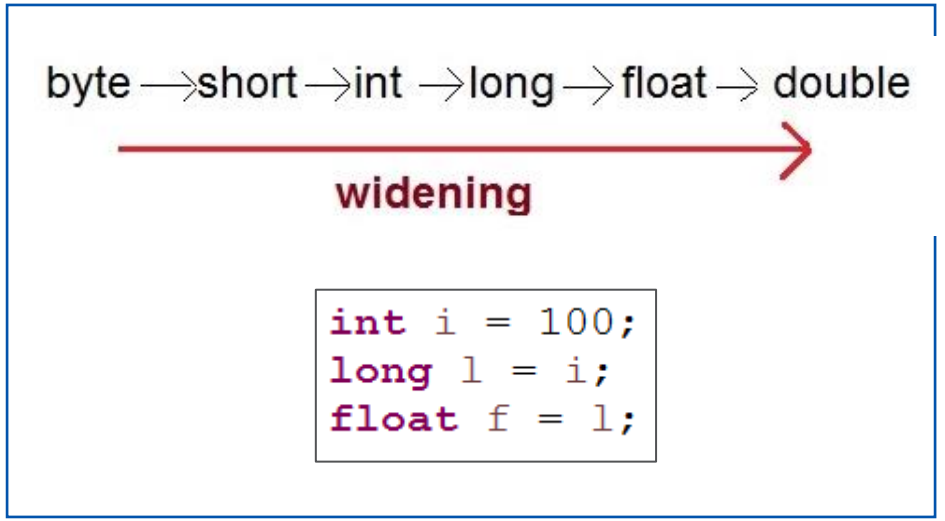
axess academy

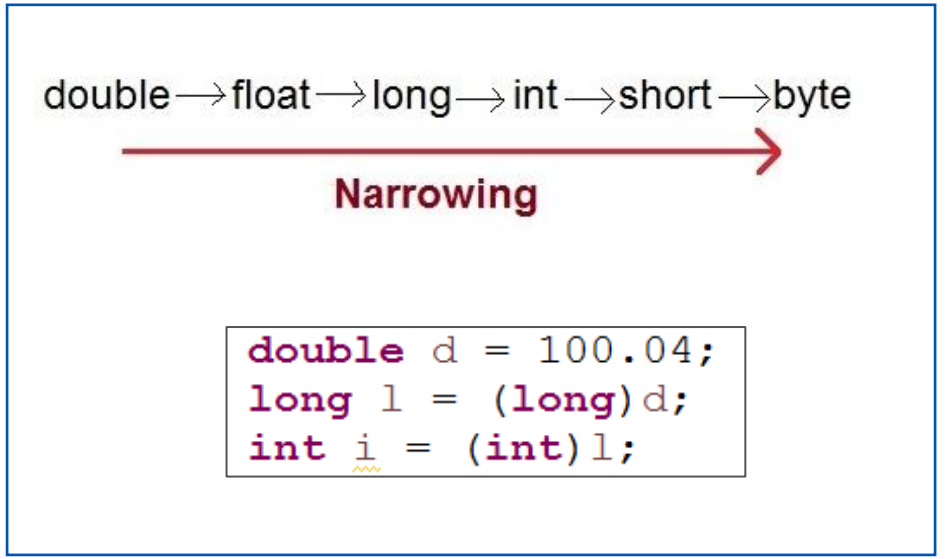
Typecasting

- Assigning a value of one type to a variable of another type is known as **Type Casting**.

- In Java, type casting is classified into two types,
  - **Widening** Casting(Implicit or automatic):
    - the two types are compatible
    - the target type is larger than the source type

  - **Narrowing** Casting(Explicitly done):
    - When you are assigning a larger type value to a variable of smaller type, then you need to perform explicit type casting.





axess academy

Operators

Operators

- Java provides a rich set of the operator (different type)to manipulate variables. We can divide all the Java operators into the following groups:
    1. Arithmetic Operators : +, -, *, /, %
    2. Assignment Operators : =, +=, -=, *=, /=, %=, &=, |=, ^=
    3. Relational Operators : >, <, ==, >= , <= , !=
    4. Unary Operator : ++, --, !
    5. Logical Operators : &&, ||, !
    6. Bitwise Operator : &, |, =, ^, >>, <<
    7. Ternary Operator: ?:

axess academy

Operators with examples

```
int a=10;
int b=5;
System.out.println(a+b);//15
System.out.println(a-b);//5
System.out.println(a*b);//50
System.out.println(a/b);//2
System.out.println(a%b);//0
```

Arithmetic operators

```
int x=10;
System.out.println(x++);//10 (11)
System.out.println(++x);//12
System.out.println(x--);//12 (11)
System.out.println(--x);//10
```

Unary operators

```
int a=10;
a+=3;//10+3
System.out.println(a);
a-=4;//13-4
System.out.println(a);
a*=2;//9*2
System.out.println(a);
a/=2;//18/2
System.out.println(a);
```

Assignment operators

```
int a=10;
int b=5;
int c=20;
boolean flag=false;
if(!flag) {
    //false && true
    System.out.println(a<b&&a<c);//false
    //false || true
    System.out.println(a<b||a<c);//true
}
```

Relational and Logical operators

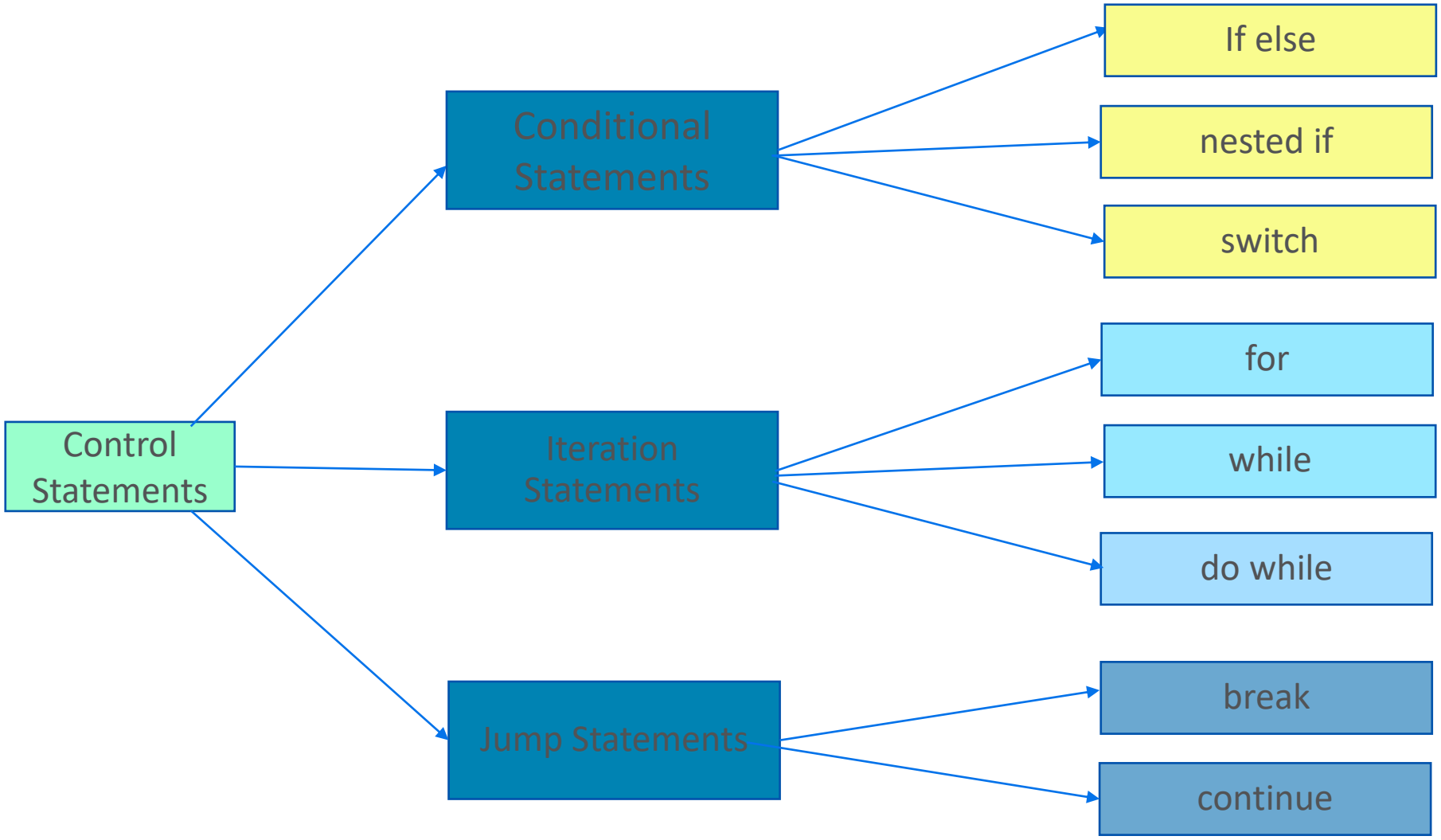
Ternary operator

```
int a=10;
int b=5;
int min=(a<b)?a:b;
System.out.println(min);
```

axess academy

Control Statements

Control Statements



axess academy

Conditional Statements

```
int number=-13;
if(number>0){
System.out.println("POSITIVE");
}else if(number<0){
System.out.println("NEGATIVE");
}else{
System.out.println("ZERO");
}
```

if-else

```
//Creating two variables for age and weight
int age=20;
int weight=80;
//applying condition on age and weight
if(age>=18){
    if(weight>50){
        System.out.println("You are eligible to donate blood");
    }
}
```

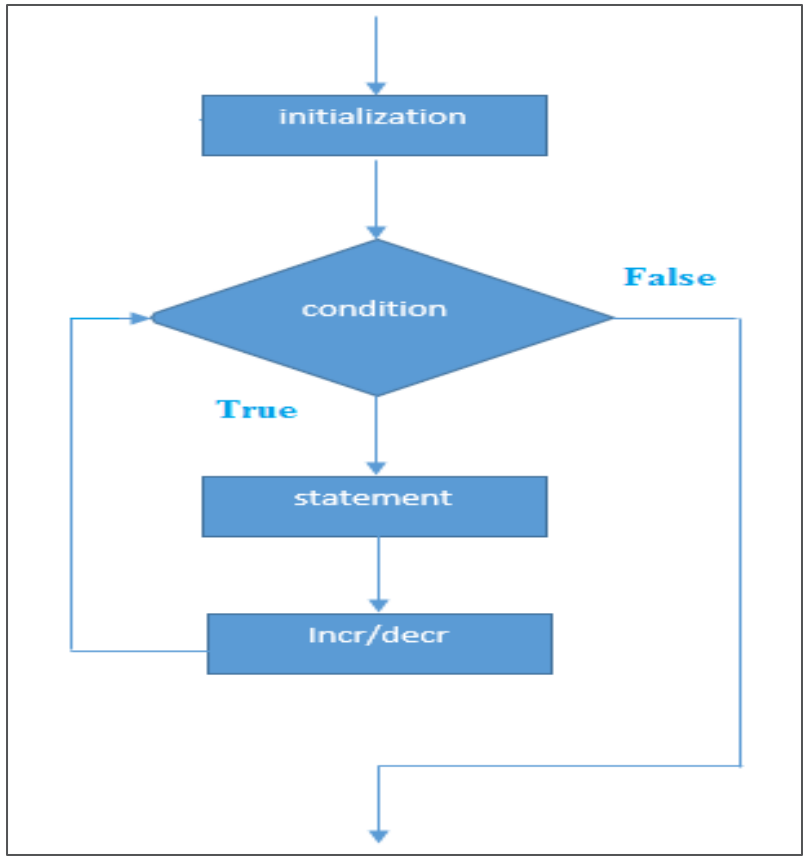
Nested if

```
Integer age = 18;
switch (age)
{
    case (16):
        System.out.println("You are under 18.");
        break;
    case (18):
        System.out.println("You are eligible for vote.");
        break;
    case (65):
        System.out.println("You are senior citizen.");
        break;
    default:
        System.out.println("Please give the valid age.");
        break;
}
```

switch

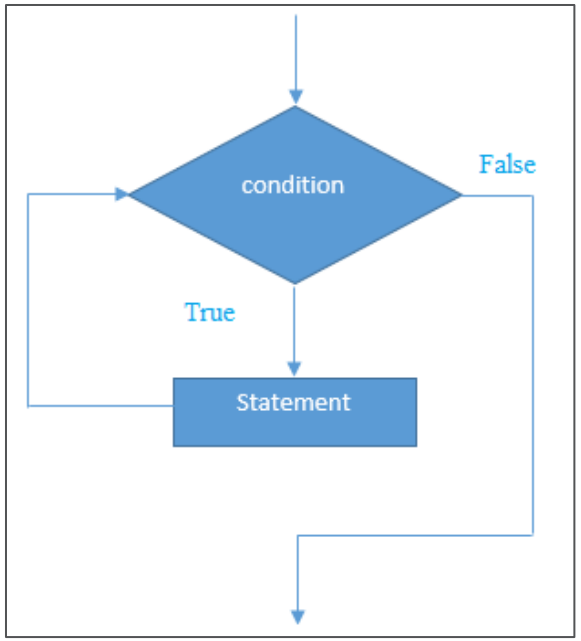
axess academy

Iteration Statements



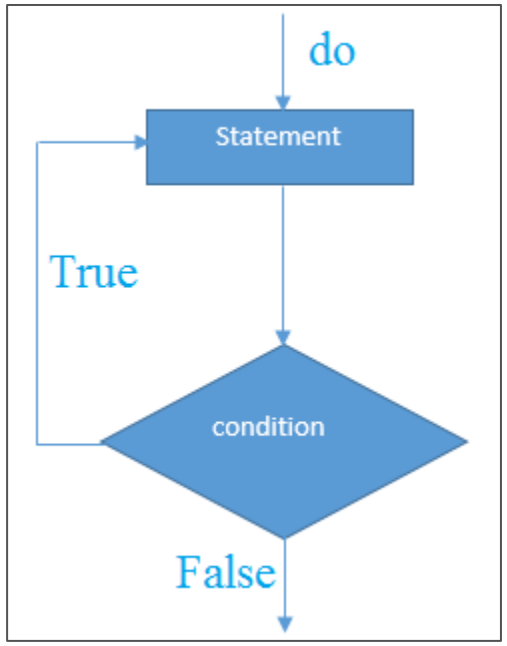
for loop

```
//Code of Java for loop
for(int i=1;i<=10;i++){
    System.out.println(i);
}
```



while loop

```
int i=1;
while(i<=10){
    System.out.println(i);
i++;
}
```



do while loop

```
int i=1;
do{
    System.out.println(i);
i++;
}while(i<=10);
```

axess academy

Jump Statements

- **break**:
  - When a break statement is encountered inside a loop, the loop is immediately terminated and the program control resumes at the next statement following the loop.
  - The Java *break* is used to break loop or switch statement.

- **continue**:
  - The continue statement is used in loop control structure when you need to jump to the next iteration of the loop immediately. It can be used with for loop or while loop.
  - The Java *continue statement* is used to continue the loop.

```
//using for loop
for(int i=1;i<=10;i++){
    if(i==5){
        //breaking the loop
        break;
    }
    System.out.println(i);
}
```

```
//for loop
for(int i=1;i<=10;i++){
    if(i==5){
        //using continue statement
        continue;//it will skip the rest statement
    }
    System.out.println(i);
}
```

Array

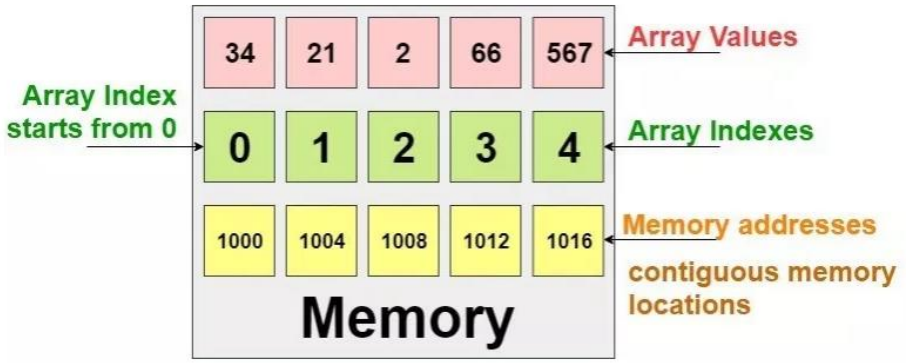
axess academy

Array

- Array in Java are a group of like-typed variables that are referred to by a common name. Array is a collection of homogenous elements that have **contiguous** memory location.
- The **size** of an array must be specified by an **int** value and not long or short.
- Array can contain primitive data types as well as objects of a class depending on the definition of array.
- In case of primitive data types, the actual values are stored in **contiguous** memory locations.
- In case of objects of a class, the actual objects are stored in heap area.

```
int arr1[]; //declaring an array
arr1=new int[5]; //allocating memory to an array
arr1[0]=10; //assigning the values
arr1[1]=20;
arr1[2]=30;
arr1[3]=40;
arr1[4]=50;

//declaring and allocating memory
int arr2[]= new int[5];

//declaring and allocating memory
int[] arr3= new int[5];

//declaring,allocating memory and assigning values
int arr4[]= {10,20,30,40,50};
```

int x[ ] = new int[ ] {34, 21, 2, 66, 567};



axess academy

Iteration of an array

- To iterate an array, we use for loop or for-each loop also.
- The for-each loop is introduced in Java5. It is mainly used to traverse array or collection elements.
- The advantage of for-each loop is that it eliminates the possibility of bugs and makes the code more readable.

```
//traditional for loop
int arr[]= {10,20,30,40,50};
for(int i=0;i<arr.length;i++) {
    System.out.println(i); //10,20,30,40,50
}
//for-each loop
for(int i: arr)
    System.out.println(i);//10,20,30,40,50
```

axess academy

Module 1 Hands-on

axess academy

Module 1 Hands-on

- Write a program to display table of 10.
- Write a program to display Fibonacci series of 10 numbers (0,0,1,1,2,3,5,8,13,21).
- Write a program to check whether a given number is odd or even.
- Write a program to find largest of two numbers.
- Write a java program to create an array of numbers and display the original array in sorted order.
- Write a java program to create an array of numbers and display the original array in reverse order.

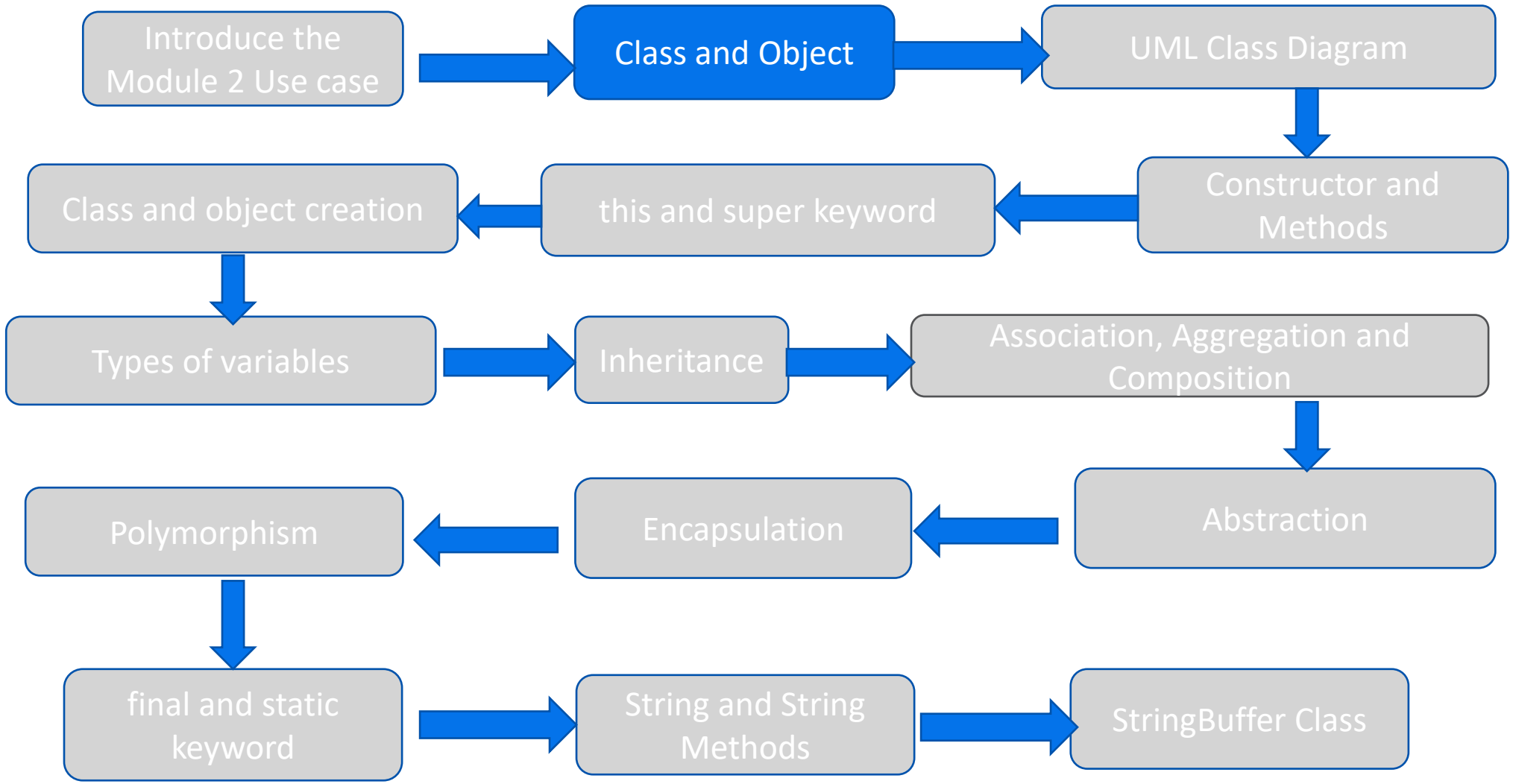
axess academy

Module 2

OOP’s programming

Module 2 : OOP's programming design

axess academy



Presenting the use case

axess academy

Bank Customer Management System

- Create the UML class diagram for customer.
- Display the menu to user.
- Create the customer class having fields cutsomerid as int, name as string, mailid as string contact as string and account type as string.
- If user enters 1 , please provide the details. On valid entries these details should be added into the array with auto generated customerid.

```
Welcome to Standard Chartered Bank
Please enter your choice
1 for Add new Customer
2 for Display Customers
3 for Search Customer
4 for Delete Customer
5 for Exit the bank application
```

```
Please enter customer details :
Enter name :
Sandra
Enter email :
Sandra@gmail.com
Enter contact :
9988778877
Enter account type (Savings or Current):
Savings
Customer added suucessfully with customer id 4656
```

axess academy

Bank Customer Management System …

- If user enters 2 , display all existing customer details.

```
Customer Id = 7262, Customer name = Sandra, Customer email = Sandra@gmail.com, Customer conta
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer c
Customer Id = 2680, Customer name = Steve, Customer email = Steve@gmail.com, Customer contact
```

- If user enters 3 , display the matched customer details.

```
3
Please enter customer id
1014
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer
```

- If user enters 5, exit the application.

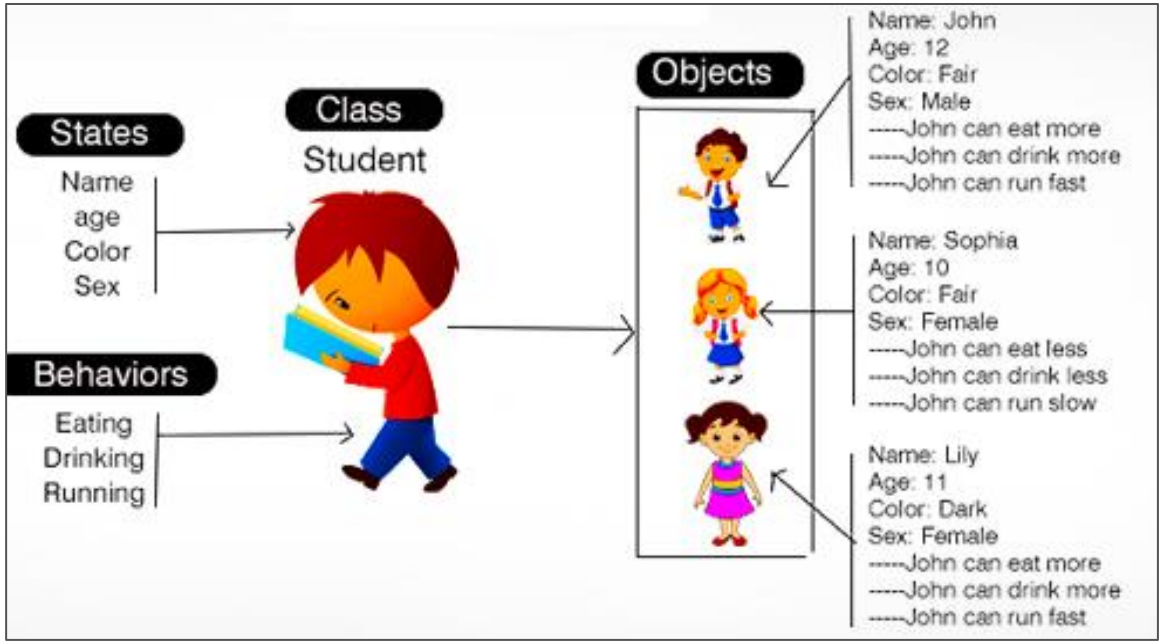
axess academy

Class and Object

axess academy

Class and Object

- Class is a template or blueprint from which **objects** are created.
- Class is a **logical** entity. It can't be **physical**.
- Any entity that has state and behaviour is known as an object.
- For example: chair, pen, table, keyboard, bike, software etc. It can be physical and logical.
  - **state:** represents **data** (value) of an object.
  - **behavior:** represents the **behavior** (functionality) of an object such as eating, drinking etc.



Constructor and Method

axess academy

Constructor

- Constructors are used to initialize the object's state. Like methods, a constructor also contains **collection of statements(i.e. instructions)** that are executed at time of Object creation.

- Constructors are used to assign values to the instance variables at the time of object creation, either explicitly done by the programmer or by Java itself (default constructor).

- Each time an object is created using **new()** keyword at least one constructor (it could be default constructor) is invoked to assign initial values to the **data members** of the same class.

- There are two type of constructor in Java:
  - **No-argument constructor** : A constructor is called "Default Constructor" when it doesn't have any parameter.
  - **Parametrized constructor** : A constructor which has a specific number of parameters is called a parameterized constructor.

axess academy

Default and Parametrized Constructor example

```
class Employee {
    /* instance varibale */
    int id;
    String name;
    float salary;
    void display() {
        System.out.println(this.id + " " + this.name + " " + this.salary);
    }
}
public class Test {
    public static void main(String args[]) {
        //Object creation using default constructor
        Employee emp1= new Employee();
        emp1.display();

    }
}
```

Default Constructor Output

```
0 null 0.0
```

```
class Employee {
    /* instance varibale */
    int id;
    String name;
    float salary;
    // Parametrized constructor
    Employee(int id, String name, float salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    void display() {
        System.out.println(this.id + " " + this.name + " " + this.salary);
    }
}
public class Test {
    public static void main(String args[]) {
        //Object creation
        Employee emp1 = new Employee(101,"John",20000);
        emp1.display();
        Employee emp2 = new Employee(102,"Jack",10000);
        emp2.display();
    }
```

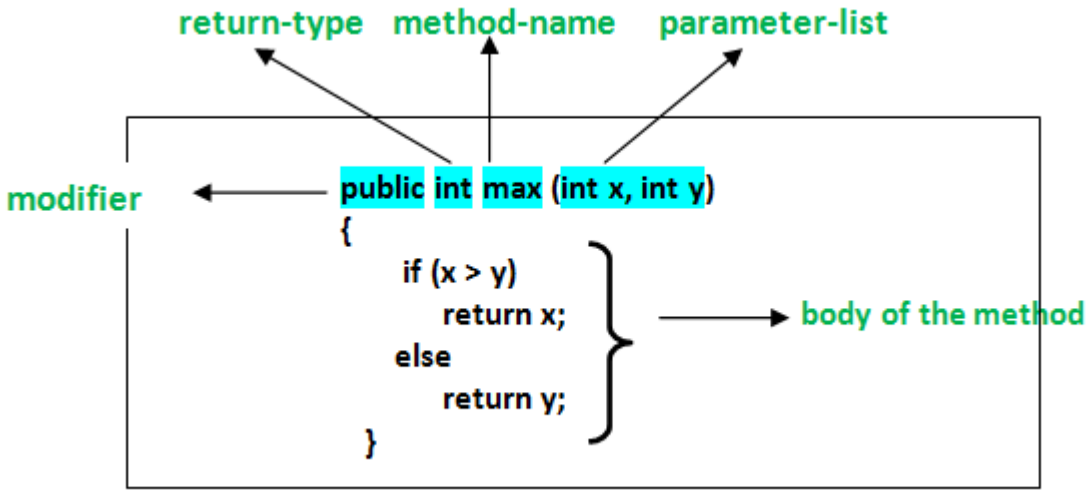
Parametrized Constructor Output

```
101 John 20000.0
102 Jack 10000.0
```

axess academy

Method in java

- A method is a collection of statements that perform some specific task and return result to the caller.
- A method can perform some specific task without returning anything.
- Methods allow us to **reuse** the code without retyping the code.



axess academy

Constructor vs method

Java Constructor	Java Method
A constructor is used to initialize the state of an object.	A method is used to expose the behavior of an object.
A constructor must not have a return type.	A method must have a return type.
The constructor is invoked implicitly.	The method is invoked explicitly.
The Java compiler provides a default constructor if you don't have any constructor in a class.	The method is not provided by the compiler in any case.
The constructor name must be same as the class name.	The method name may or may not be same as the class name.
Constructor overloading is supported.	Method overloading is supported.

this

axess academy

this keyword

- There can be a lot of usage of **java this keyword**. In java, this is a **reference variable** that refers to the current object.

- Here is given the 6 usage of java this keyword.
  1. this can be used to refer current class instance variable.
  2. this() can be used to invoke current class constructor.
  3. this can be used to invoke current class method (implicitly)

```
// Parametrized constructor
Employee(int id, String name) {
    this.id = id;
    this.name = name;
}

// Parametrized constructor
Employee(int id, String name, float salary) {
    this(id,name);
    this.salary = salary;
}

void first() {
    System.out.println(this.id + " " + this.name);
    this.second();
}

void second() {
    System.out.println(this.salary);
}
```

axess academy

Class and Object creation

axess academy

Class and object creation

```java
class Employee {
    /* instance varibale */
    int id;
    String name;
    float salary;
    // Parametrized constructor
    Employee(int id, String name, float salary) {
        this.id = id;
        this.name = name;
        this.salary = salary;
    }
    void display() {
        System.out.println(this.id + " " + this.name + " " + this.salary);
    }
}
```

Employee class

```
101 John 20000.0
102 Jack 10000.0
```

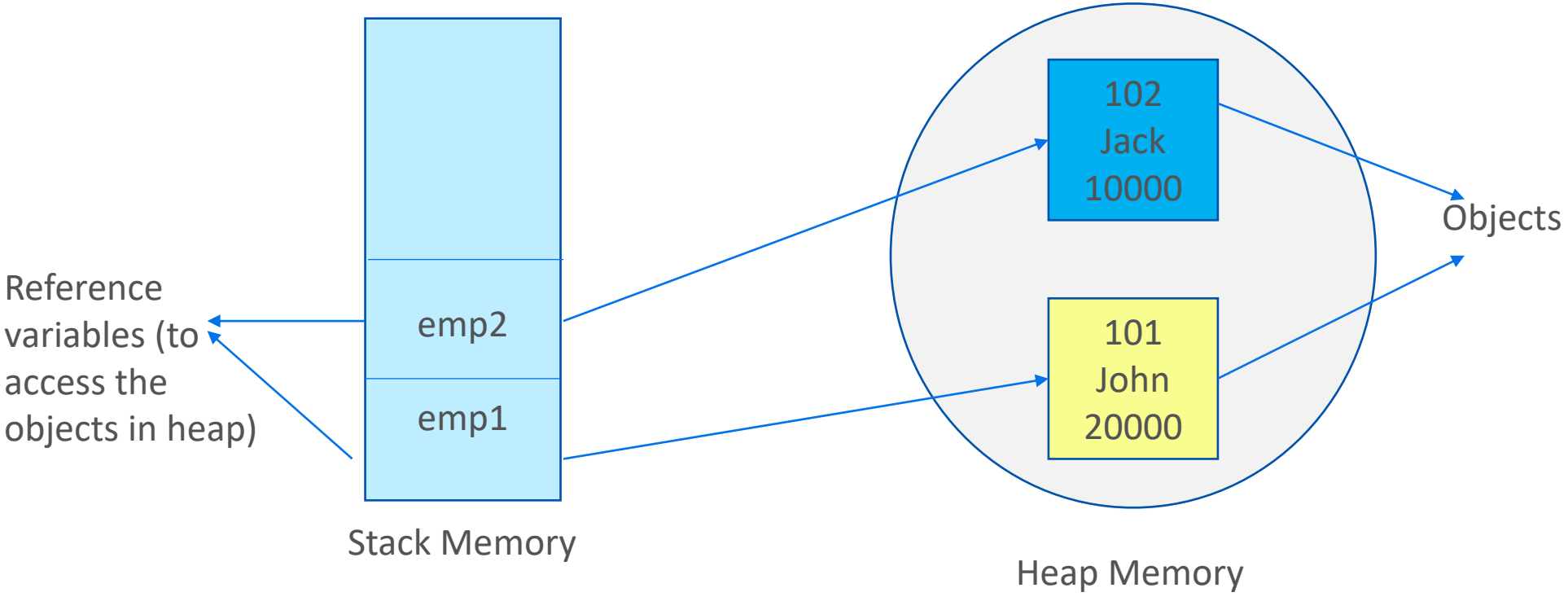
Output

```java
public class Test {
    public static void main(String args[]) {
        //Object creation
        Employee emp1 = new Employee(101,"John",20000);
        emp1.display();
        Employee emp2 = new Employee(102,"Jack",10000);
        emp2.display();
    }
}
```

main class creates the objects

axess academy

How the objects gets stored



Types of variables

axess academy

Types of variables

- **Local Variables** :
  - A variable defined within a block or method or constructor is called local variable. These variables are created when the block in entered or the function is called and destroyed after exiting from the block or when the call returns from the function.
  - The scope of these variables exists only within the block in which the variable is declared. i.e. we can access these variable only within that block.
  - Initialization of Local Variable is Mandatory.

- **Instance Variables**:
  - Instance variables are non-static variables and are declared in a class outside any method, constructor or block. As instance variables are declared in a class, these variables are created when an object of the class is created and destroyed when the object is destroyed.
  - Unlike local variables, we may use access specifiers for instance variables. If we do not specify any access specifier then the default access specifier will be used.
  - Initialization of Instance Variable is not Mandatory.
  - Instance Variable can be accessed only by creating objects.

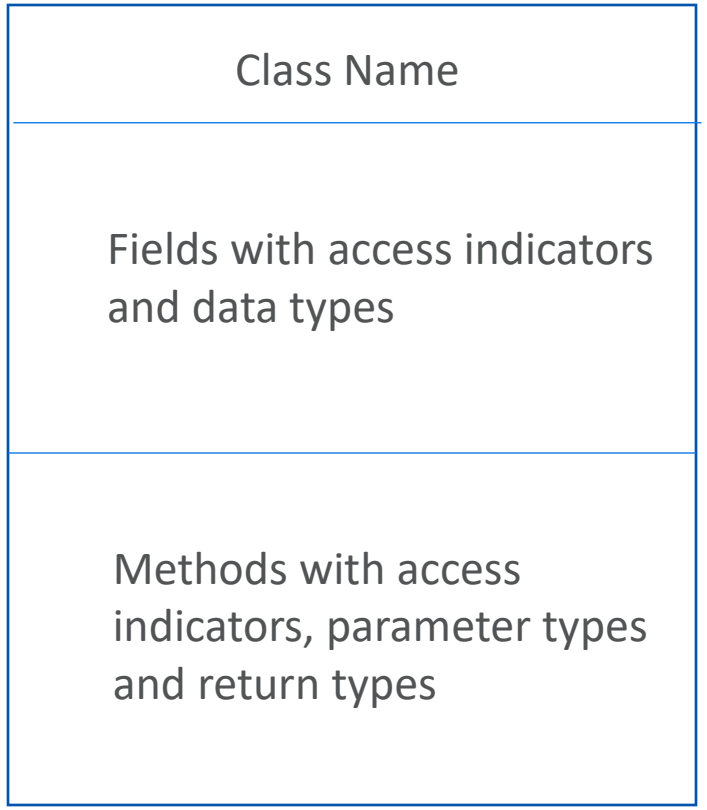
- **Static Variables**:
  - We will see in upcoming slides

axess academy

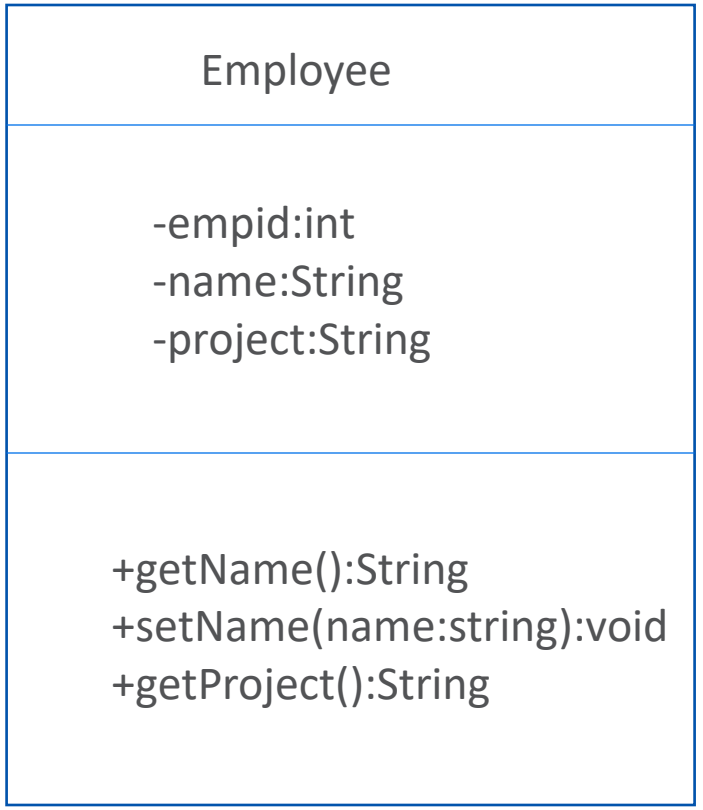
UML class diagram

axess academy

UML Class Diagram



Minus (-) for Private
Plus (+) for Public
Pound (#) for Protected



Employee class

axess academy

How to take user input

axess academy

Scanner

- Scanner is a class in java.util package used for obtaining the input of the primitive types like int, double, etc. and strings. It is the easiest way to take user input in a Java program.
- It has different set of methods for different data types like next(for String), nextInt(for int)

```
// Declare the object and initialize with
// predefined standard input object
Scanner sc = new Scanner(System.in);

// String input
String name = sc.next();

// Character input
char gender = sc.next().charAt(0);

// Numerical data input
// byte, short and float can be read
// using similar-named functions.
int age = sc.nextInt();
long mobileNo = sc.nextLong();
double cgpa = sc.nextDouble();

// Print the values to check if the input was correctly obtained.
System.out.println("Name: "+name);
System.out.println("Gender: "+gender);
System.out.println("Age: "+age);
System.out.println("Mobile Number: "+mobileNo);
System.out.println("CGPA: "+cgpa);
```

Let us create class and object for the use case

axess academy

Use Case on class and object creation

- Create the UML class diagram for customer class.
- Let us create the customer class having fields cutsomerid as int, name as string, mailid as string, salary as double, dept as string, location as string contact as string and account type as string.
- Create 3 customer class objects using parametrized constructor and insert them into an array of customers.
- Finally iterate the array and display all customer objects.

axess academy

Inheritance

axess academy

Inheritance

- It is the mechanism in java by which one class is allow to **inherit** the *features(fields and methods)* of another class. The idea behind inheritance in java is that you can create new classes that are built upon existing classes.

- When you inherit from an existing class, you can **reuse** methods and fields of parent class, and you can add new methods and fields also.

- The class which inherits the properties of other is known as **subclass** (derived class, child class) and the class whose properties are inherited is known as **superclass** (base class, parent class).

```
class Animal
{  void eat()
   {
      System.out.println("eating...");
   }
}
class Dog extends Animal
{  void bark()
   {
      System.out.println("barking...");
   }
}
class TestInheritance
{  public static void main(String args[])
   {
      Dog d=new Dog();
      d.bark();
      d.eat();
   }
}
```

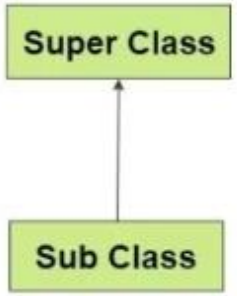
Output

```
barking...
eating...
```

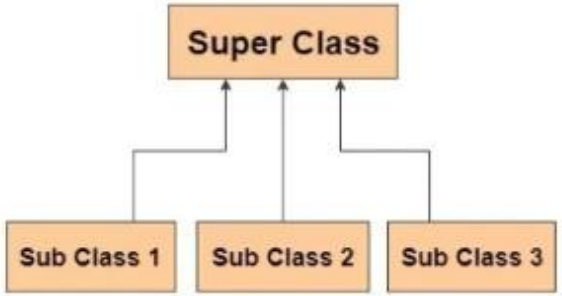
axess academy

Types of Inheritance

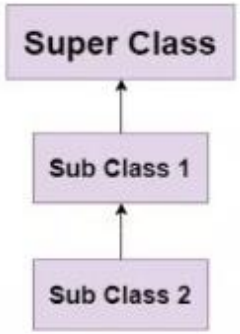
Single Inheritance



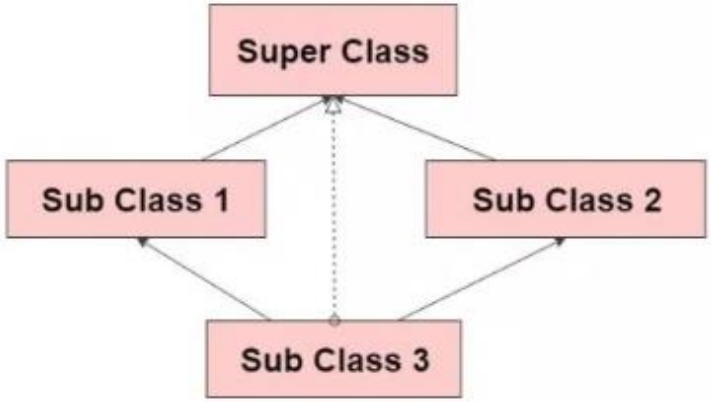
Hierarchial Inheritance



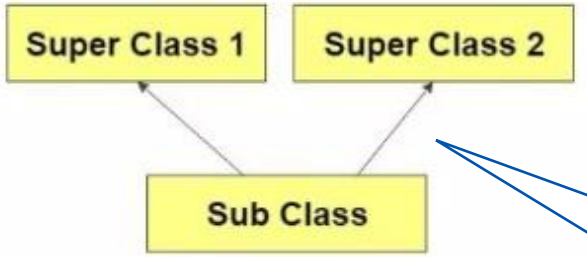
MultiLevel Inheritance



Hybrid Inheritance



Multiple Inhertance

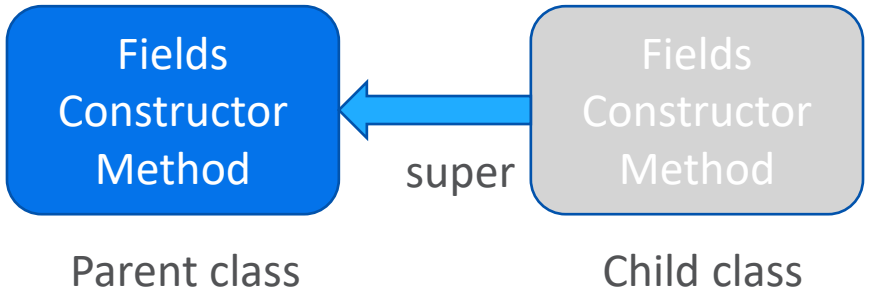


Supported using interfaces

axess academy

super keyword

- The **super** keyword in java is used to refer parent class data members.
- Using “super” keyword, you can access fields, constructor or methods of super class.



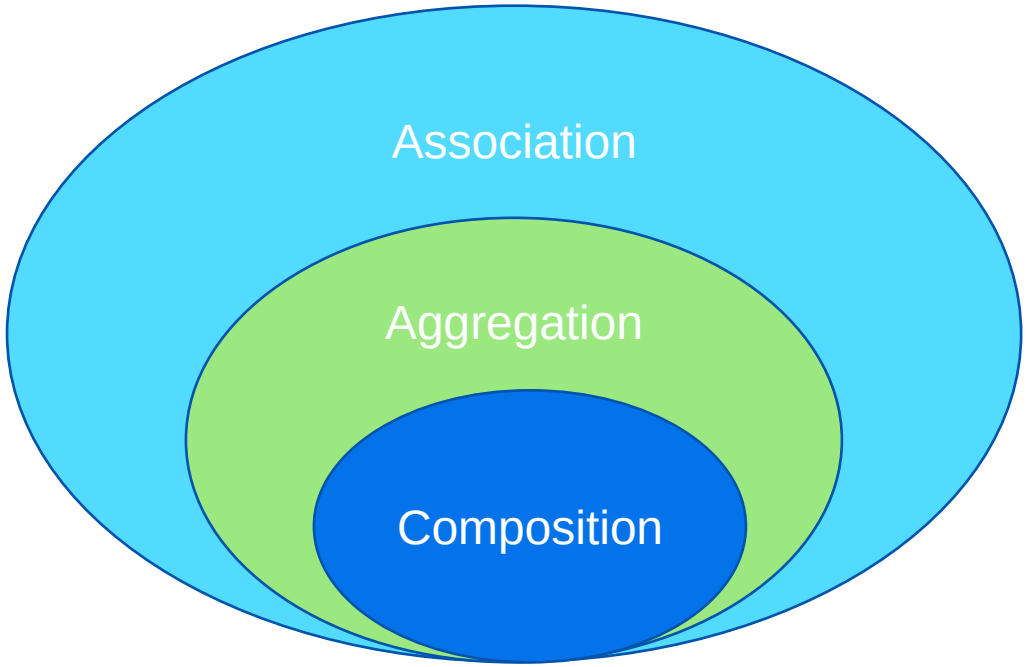

```java
/* Base class vehicle */
class Vehicle
{
    int maxSpeed = 100;
}

/* sub class Car extending vehicle */
class Car extends Vehicle
{
    int maxSpeed = 120;

    void display()
    {
        /* print maxSpeed of base class (vehicle) */
        System.out.println("Maximum Speed: " + super.maxSpeed);
    }
}
```

Association vs Aggregation vs Composition

axess academy

Association vs Aggregation vs Composition

- Association refers to the relationship between multiple objects. It refers to how objects are related to each other and how they are using each other's functionality.
- Association can be one-to-one, one-to-many, many-to-one, many-to-many.
- **Composition** and **Aggregation** are the two forms of association.



axess academy

Aggregation vs Composition

Properties	Aggregation	Composition
Dependency	An association is said to be aggregation if both Objects can exist independently. For example, a Team object and a Player object. The team contains multiple players but a player can exist without a team.	An association is said to composition if an Object owns another object and another object cannot exist without the owner object. Example: Car and Engine. Here Car object contains the engine and engine cannot exist without Car.
Type of Relationship	"has-a"	"part-of"
Type of association	Weak association	Strong association

Aggregation

```
//Player Object
    class Player {}

//Team
 class Team {
   //players can be 0 or more
   private List players;

   public Team() {
      players = new ArrayList();
   }
}
```

Composition

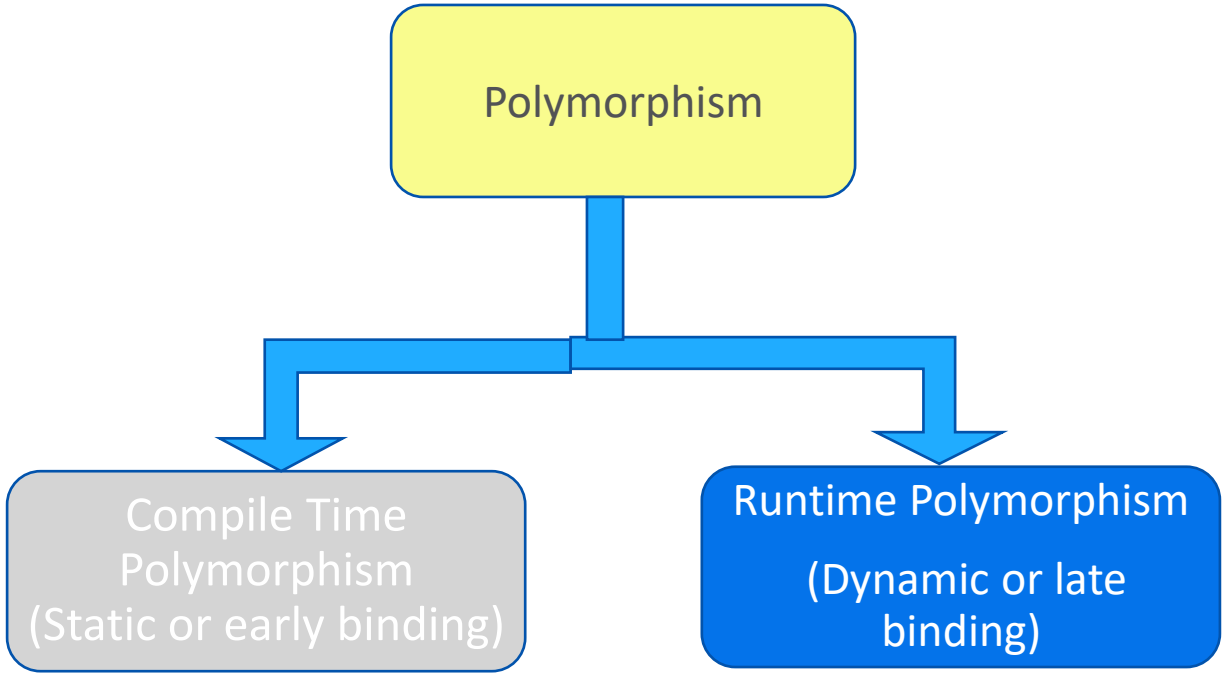
```
//Engine Object
class Engine {}
//Car must have Engine

class Car {
   //engine is a mandatory
   //part of the car
   private Engine engine;

   public Car () {
      engine = new Engine();
   }
}
```

Polymorphism

axess academy

Polymorphism

- Polymorphism refers to the ability of OOPs programming languages to differentiate between entities with the same name efficiently. It can be achieved in two ways.
- Static binding is done during compile-time while dynamic binding is done during run-time.
- private, final , static methods and variables use static binding and bonded by compiler while overridden methods are bonded during runtime based upon type of runtime object



axess academy

Compile Time Polymorphism (Method Overloading)

- Method overloading is one of the ways in which Java supports Compile Time Polymorphism.
- Two or more method in a class have same name but different parameters.
- Overloading always occur in the same class(unlike method overriding).
- Method overloading can be done by changing number of arguments or by changing the data type of arguments.
- If two or more method have same name and same parameter list **but differs in return type are not** said to be overloaded method.
- Overloaded method can have different access modifiers.

```
class Calculate {
    void sum(int a, int b) {
        System.out.println("sum is in first method =" + (a + b));
    }
    void sum(float a, float b) {
        System.out.println("sum is in second method =" + (a + b));
    }
    void sum(int a, float b, int c) {
        System.out.println("sum is in third method =" + (a + b + c));
    }
}
public class Test {
    public static void main(String args[]) {
        Calculate cal = new Calculate();
        cal.sum(8,5); // sum(int a, int b) is method is called.
        cal.sum(4.6f, 3.8f); // sum(float a, float b) is called.
        cal.sum(10,4.6f, 3); // sum(int a, float b , int c) is called.
    }
}
```

Code Snippet

```
sum is in first method =13
sum is in second method =8.4
sum is in third method =17.6
```

Output

axess academy

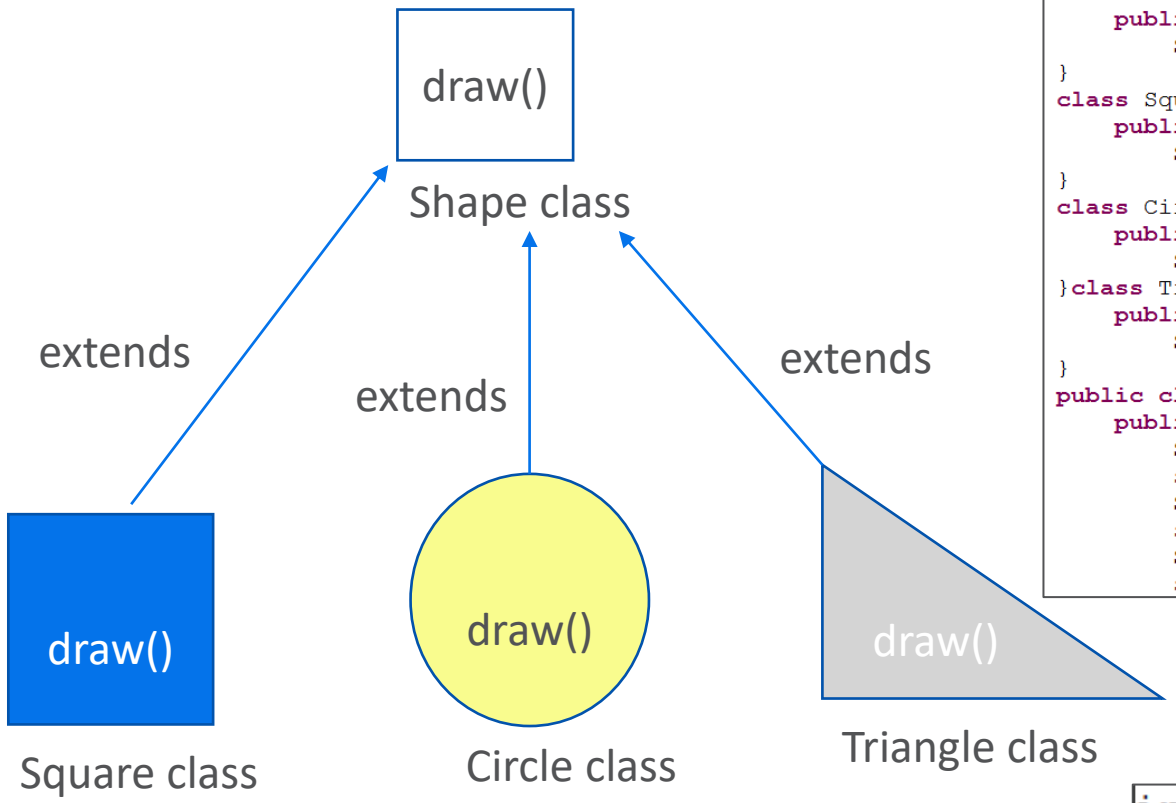
Runtime Polymorphism (Method Overriding)

- Method overriding is one of the ways in which Java supports Runtime Polymorphism. It is the mechanism by which a call to an overridden method is resolved at run time, rather than compile time.
- When an overridden method is called through a superclass reference, Java determines which version (superclass/ subclasses) of that method is to be executed based upon the type of the object being referred to at the time the call occurs. Thus, this determination is made at run time.
- At run-time, it depends on the type of the object being referred to (not the type of the reference variable) that determines which version of an overridden method will be executed.
- A superclass reference variable can refer to a subclass object. This is also known as upcasting. Java uses this fact to resolve calls to overridden methods at run time.

```
class Shape {
    public void draw() {
        System.out.println("inside Shape class draw method");}
}
class Square extends Shape {
    public void draw() {
        System.out.println("inside Square class draw method");}
}
class Circle extends Shape {
    public void draw() {
        System.out.println("inside Circle class draw method");}
}class Triangle extends Shape {
    public void draw() {
        System.out.println("inside Triangle class draw method");}
}
public class Test {
    public static void main(String args[]) {
        Shape s1 = new Square();
        s1.draw();
        Shape s2 = new Circle();
        s2.draw();
        Shape s3 = new Triangle();
        s3.draw();
```

axess academy

Method Overriding Diagram




```
class Shape {
    public void draw() {
        System.out.println("inside Shape class draw method");}
}
class Square extends Shape {
    public void draw() {
        System.out.println("inside Square class draw method");}
}
class Circle extends Shape {
    public void draw() {
        System.out.println("inside Circle class draw method");}
}class Triangle extends Shape {
    public void draw() {
        System.out.println("inside Triangle class draw method");}
}
public class Test {
    public static void main(String args[]) {
        Shape s1 = new Square();
        s1.draw();
        Shape s2 = new Circle();
        s2.draw();
        Shape s3 = new Triangle();
        s3.draw();
```

Code Snippet

```
inside Square class draw method
inside Circle class draw method
inside Triangle class draw method
```

Output

axess academy

Method Overloading vs Method Overriding

Method Overloading	Method Overriding
Parameter must be different and name must be same.	Both name and parameter must be same.
Compile time polymorphism.	Runtime polymorphism.
Increase readability of code.	Increase reusability of code.
Access specifier can be changed.	Access specifier cannot be more restrictive than original method(can be less restrictive).

Encapsulation

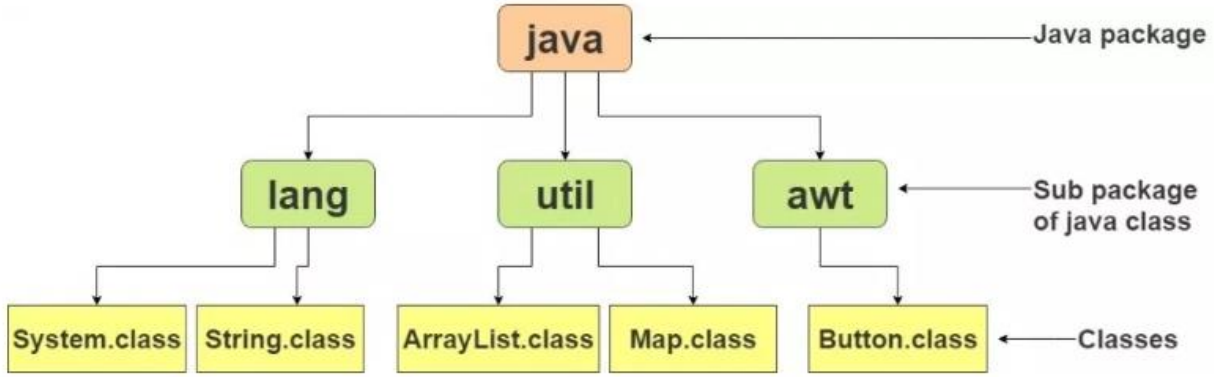
axess academy

Packages

- Package in java as the name suggests is a **pack(group) of classes, interfaces and other packages**. They can be considered as data encapsulation (or data-hiding). Packages are used for below points:
  - Preventing naming conflicts. For example there can be two classes with name Employee in two packages, college.staff.cse.Employee and college.staff.ee.Employee
  - Providing controlled access: protected and default have package level access control. A protected member is accessible by classes in the same package and its subclasses. A default member (without any access specifier) is accessible by classes in the same package only.

- There are two types of packages **: In-built** (java.util, java.lang etc.) and **User Defined**.
- Syntax to access the classes, interfaces inside the package:

```
import java.util.*;
```

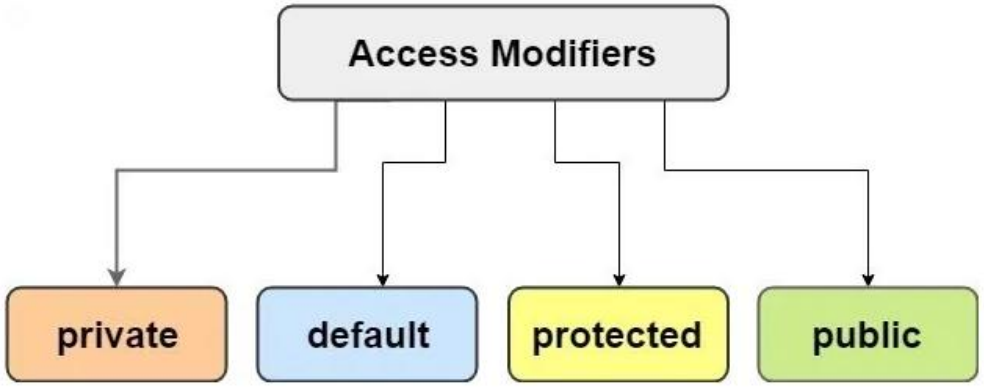


In-Built Packages

axess academy

Access Modifiers

- Access modifiers in java specify the scope of a class, constructor , variable , method or data member. There are four types of access modifiers available in java:
  1. **Private**
  2. **Default –** *No keyword required*
  3. **Protected**
  4. **Public**



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

Let us implement encapsulation

axess academy

Encapsulation implementation

- Make all the customer fields as private. Write getter and setters for customer class for all fields.
- Create customer objects using setter methods.
- Display the array of customer objects.

Abstraction

axess academy

Abstraction

- Data Abstraction can be defined as the process of identifying only the required characteristics of an object ignoring the irrelevant details. Example: An ATM machine.

- Abstraction can be achieved in two ways:
  - Abstract Class
  - Interface

Abstract Class

- An abstract class is a class that is declared with abstract keyword.
- An abstract class can never be instantiated. If a class is declared as abstract then the sole purpose is for the class to be extended.
- A class cannot be both abstract and final. (since a final class cannot be extended).
- An abstract method is a method that is declared without an implementation.

```
abstract class Shape
{
    String color;

    // these are abstract methods
    abstract double area();
    // abstract class can have constructor
    public Shape(String color) {
        System.out.println("Shape constructor called");
        this.color = color;
    }
    // this is a concrete method
    public String getColor() {
        return color;
    }
}
```

```
class Circle extends Shape
{
    double radius;
    public Circle(String color,double radius) {
        // calling Shape constructor
        super(color);
        System.out.println("Circle constructor called");
        this.radius = radius;
    }
    @Override
    double area() {
        return Math.PI * Math.pow(radius, 2);
    }
}
```

axess academy

Interface

- Like a class, an interface can have methods and variables.
- All the methods are public and abstract. And all the fields are public, static, and final.
- Interfaces specify what a class must do and not how. It is the blueprint of the class.
- If a class implements an interface and does not provide method bodies for all functions specified in the interface, then class must be declared abstract.
- We can’t create instance(interface can’t be instantiated) of interface but we can make reference of it that refers to the Object of its implementing class.
- A class can implement more than one interface.

```
interface test
{
    // public, static and final
    final int a = 10;

    // public and abstract
    void display();
}

// A class that implements interface.
class testClass implements test
{
    // Implementing the capabilities of
    // interface.
    public void display()
    {
        System.out.println("interface example");
    }
}
```

Keywords: final, static

axess academy

final keyword

The **final keyword** in java is used to restrict the user. The java final keyword can be used in many context. Final can be:

final variable → Stop value change →

```
final String companyName="java";
```

final method → Prevent method overriding →

```
final void display() {
    System.out.println("Hello final method");
}
```

final class → Prevent inheritance →

```
final class Employee {
```

axess academy

static keyword

- The **static keyword** in Java is used for memory management mainly. We can apply java static keyword with variables, methods, blocks and nested class. The static keyword belongs to the class than an instance of the class.
- The static can be:
  1. Variable (also known as a class variable) : shared to all objects
  2. Method (also known as a class method) : no need to create object to call method and static methods can not be overridden because static method is bound to class whereas method overriding is associated with object i.e. at runtime.
  3. Block : used to initialize the static data member and executed before the main method gets executed.

static variable

static variable (class variable) → Common property for all objects → `static int counter=0;`

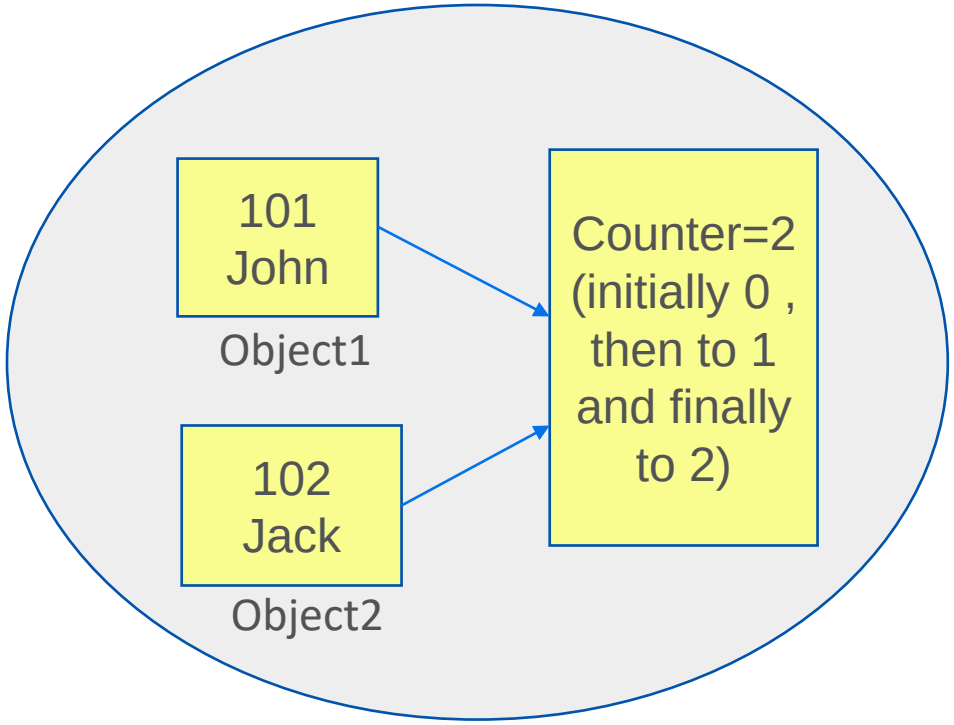
```
class Employee {
     int id;
     String name;
     static int counter = 0;
     Employee(int id, String name) {
          this.id = id;
          this.name = name;
          counter++;
     }
     void display() {
          System.out.println("Total number of objects are " + counter);
     }}
public class Test {
     public static void main(String args[]) {
          Employee emp1 = new Employee(101, "John");
          emp1.display();

          Employee emp2 = new Employee(102, "Jack");
          emp2.display();
     }}
```

Output

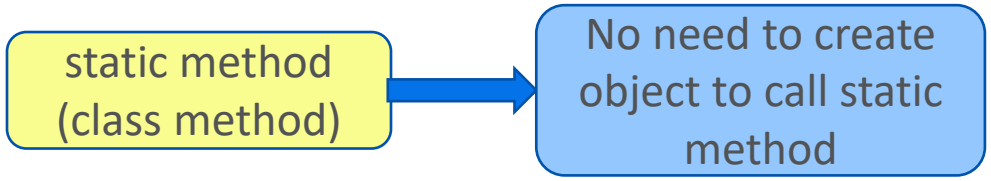
```
Total number of objects are 1
Total number of objects are 2
```

axess academy

How static variable gets stored in heap

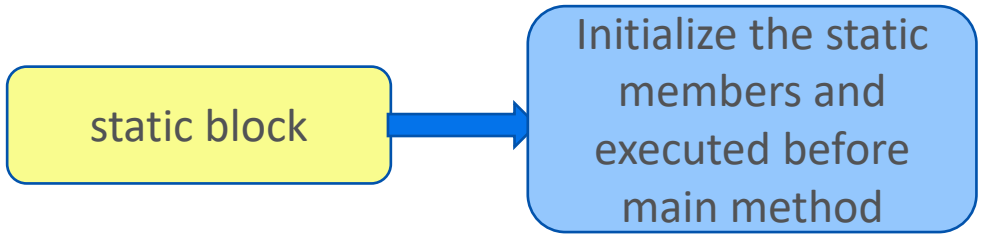


static method




```
class Calculate {
    static int cube(int x) {
        return x * x * x;
    }
}
class CalculateDemo{
    public static void main(String args[]) {
        int result = Calculate.cube(5);
        System.out.println(result);
    }
}
```

static block




```
class Employee {
    int id;
    String name;
    static int counter;
    static{
        System.out.println("static block is invoked");
        counter=0;
    }
    Employee(int id, String name) {
        this.id = id;
        this.name = name;
        counter++;
    }
    void display() {
        System.out.println("Total number of objects are " + counter);
    }}
public class Test {
    public static void main(String args[]) {
        Employee emp1 = new Employee(101, "John");
        emp1.display();
        Employee emp2 = new Employee(102, "Jack");
        emp2.display();
    }}
```

axess academy

Module 3

Strings & Exception Handling

String

String

- String is an object that represents a sequence of characters. The java.lang.String class is used to create a string object. String objects are immutable (Once you create you can not change it.)
- There are two ways to create String object:
  - **By string literal:**

    String s="Welcome";

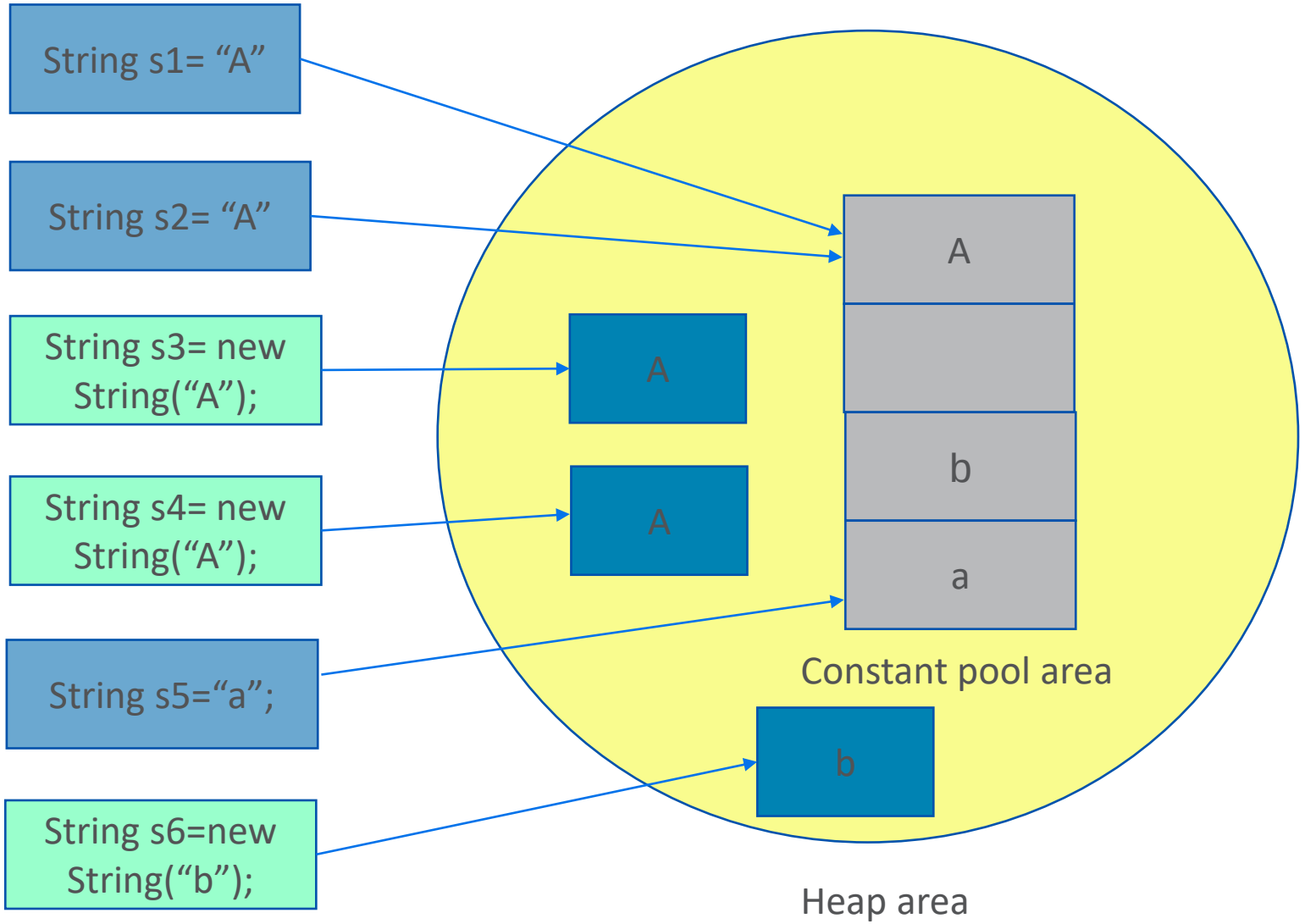
    (creates one string in constant pool area if not found else it will point to same string)
  - **By new keyword**:

    String s= new String("Welcome");

    (creates one string in constant pool area and one object in heap area)

axess academy

How String gets stored



s1.equals(s2)=>true
s1==s2=> true
s3.equals(s4)=>true
s3==s4=> false

//equals method compares the content of strings whereas == compares the memory references of variables

axess academy

String Methods

Method	Description	Code
char charAt(int index)	returns char value for the particular index	“company”.charAt(2) =>”m”
int length()	returns string length	“company”.length() => 7
String substring(int beginIndex)	returns substring for given begin index.	“company”.substring(2)=> “mpany”
String substring(int beginIndex, int endIndex)	returns substring for given begin index and end index.	“company”.substring(2,5)=> ”mpa”
boolean contains(CharSequence s)	returns true or false after matching the sequence of char value.	“company”.contains(“mp”)=> true
static String equalsIgnoreCase(String another)	compares another string. It doesn't check case.	“company”.equalsIgnoreCase(“COMpANy”)=> true
boolean equals(Object another)	checks the equality of string with the given object.	“company”.equals(“Company”)=>false

String Methods cont…

Method	Description	code
boolean isEmpty()	checks if string is empty.	“”.isEmpty()=> true
String concat(String str)	concatenates the specified string.	“company”.concat(“ name”)=> “company name”
String replace(char old, char new)	replaces all occurrences of the specified char value.	“company”.replace(“com”,”tom”)=> “tompany”
int indexOf(String str)	returns the specified char value index.	“company”.indexOf(“m”)=>2
String toLowerCase()	returns a string in lowercase.	“company”.toLowerCase()=>”company”
String toUpperCase()	returns a string in uppercase.	“company”.toUpperCase()=>”COMPANY”
String trim()	removes beginning and ending spaces of this string.	“ company”.trim() =>”company”
boolean matches(String regex)	Returns true if it matches with regex	“company”.matches(“[a-z]+”) => true
String[] split(String str)	Returns an array of strings after match of given expression	“company:name”.split(“:”)=>[“company”,”name”]

axess academy

Regular Expression Pattern and Quantifiers

Pattern	Description
[0-9]	Find any of the digits between the brackets
[A-Z]	Find any of the characters (capital letters) between the brackets
[0-9a-z]	Find any of the alphanumeric characters between the brackets
(x|y)	Find any of the alternatives separated with |

Quantifier	Description
[0-9]+	Matches any string that contains at least one *digit*
[0-9]*	Matches any string that contains zero or more occurrences of *n*
[0-9]?	Matches any string that contains zero or one occurrences of *n*
[0-9]{3}	Matches any string that contains exactly 3 occurrences of *n*
[0,9]{3,6}	Matches any string that contains minimum 3 occurrences of *n and max 6 occurrences*

- **Example 1**: To check name field can only have alphabets.
- The pattern can be :
  - /^[A-Za-z]+$/ or
  - “^[A-Za-z]+$”
  - /^[a-z]+$/i (case insensitive)
- **Example 2**: To check contact field can only have numbers of 10 digits.
- The pattern can be :
  - /^[7-9]{1}[0-9]{9}$/ or
  - “^[7-9]{1}[0-9]{9}$”

axess academy

StringBuffer

- A string buffer is like a String, but mutable i.e. can be modified.
- It contains some particular sequence of characters, but the length and content of the sequence can be changed through in-built methods.
- Every string buffer has a capacity , the default capacity is 16 chars.

```
StringBuffer sb = new StringBuffer("Welcome all");
sb.append(" to bank");
System.out.println(sb); // Welcome all to bank
```

axess academy

Annotations

- Java **Annotation** is a tag that represents the *metadata* i.e. attached with class, interface, methods or fields to indicate some additional information which can be used by java compiler and JVM.
- Annotations in java are used to provide additional information.
- Built-In Java Annotations used in java code
  - @Override :
  - @SuppressWarnings
  - @Deprecated

```
class Animal{
void eatSomething(){System.out.println("eating something");}
}

class Dog extends Animal{
@Override
void eatsomething(){System.out.println("eating foods");}//should be eatSomething
}
```

Annotations in detail

- **@Override** annotation assures that the subclass method is overriding the parent class method. If it is not so, compile time error occurs. Sometimes, we does the silly mistake such as spelling mistakes etc. So, it is better to mark @Override annotation that provides assurance that method is overridden.

- **@SuppressWarnings** annotation: is used to suppress warnings issued by the compiler.

- **@Deprecated** annotation : It marks that this method is deprecated so compiler prints warning. It informs user that it may be removed in the future versions. So, it is better not to use such methods.

Let us work on use case

axess academy

Bank Customer Management System

- Display the menu to user.

```
Welcome to Standard Chartered Bank
Please enter your choice
1 for Add new Customer
2 for Display Customers
3 for Search Customer
4 for Delete Customer
5 for Exit the bank application
```

- If user enters 1 , please provide the details. On valid entries these details should be added into the array with auto generated customerid.

```
Please enter customer details :
Enter name :
Sandra
Enter email :
Sandra@gmail.com
Enter contact :
9988778877
Enter account type (Savings or Current):
Savings
Customer added suucessfully with customer id 4656
```

axess academy

Bank Customer Management System …

- If user enters 2 , display all existing customer details.

```
Customer Id = 7262, Customer name = Sandra, Customer email = Sandra@gmail.com, Customer conta
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer c
Customer Id = 2680, Customer name = Steve, Customer email = Steve@gmail.com, Customer contact
```

- If user enters 3 , display the matched customer details.

```
3
Please enter customer id
1014
Customer Id = 1014, Customer name = Michelle, Customer email = Michelle@gmail.com, Customer
```

- If user enters 5, exit the application.

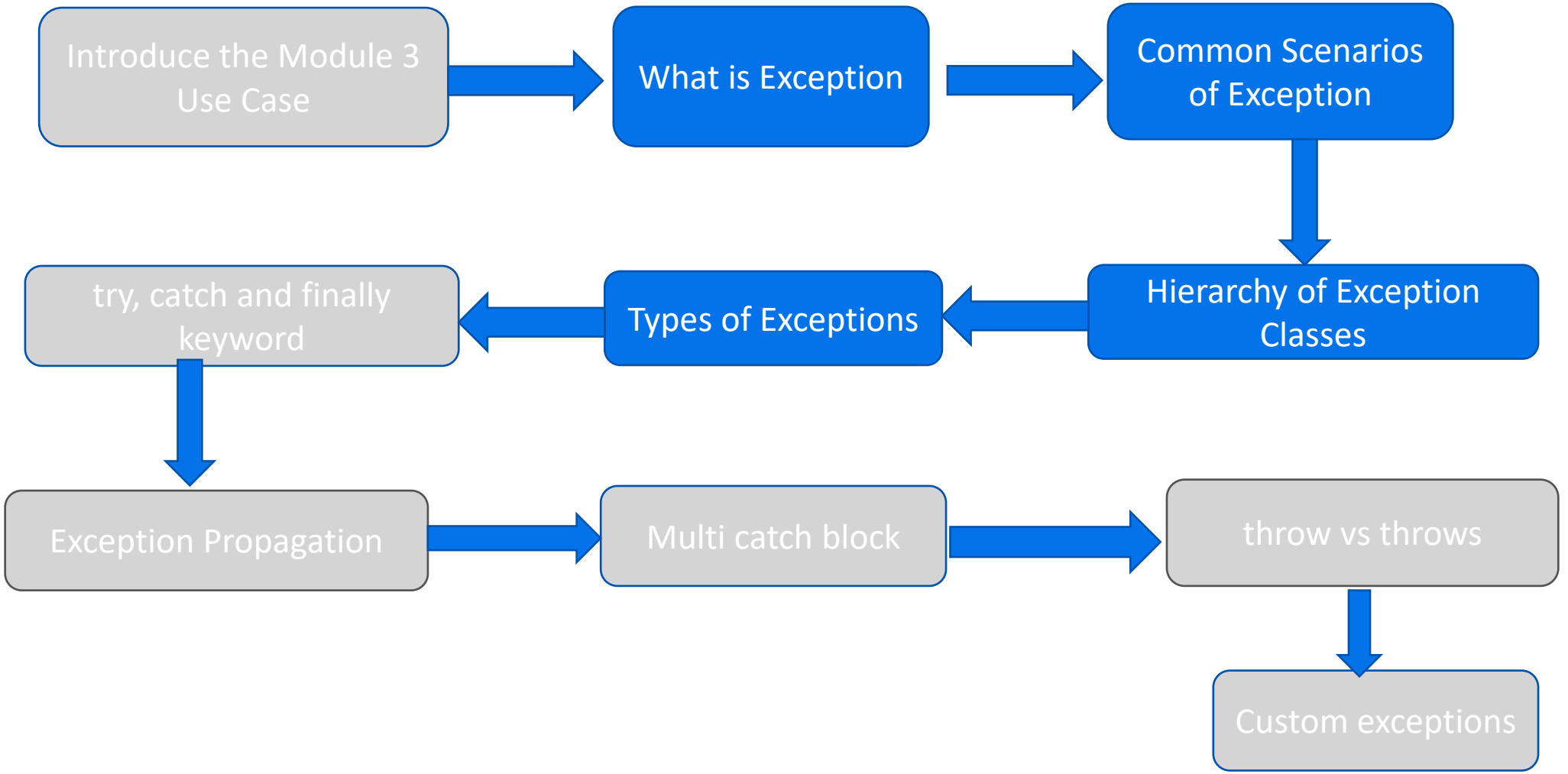
axess academy

Module 3

Exception Handling

Module 3 : Exception Handling Design

axess academy



Presenting the use case

axess academy

Bank Customer Management System

- While adding the customer, make sure admin provides valid entries like name can only have alphabets, email should be valid one , contact no should have 10 digits and account type can be either Savings or current.

- If any one of the condition fails, the customer should not be added in the array and display the appropriate error message using custom exceptions.

- Only for valid entries, it should be added in array.

- While taking the input from the user to search a customer using customer id, make sure it has to be an integer else display the appropriate error message using custom exceptions.

```
Please enter customer details :
Enter name :
32d328
Name can only have alphabets
```

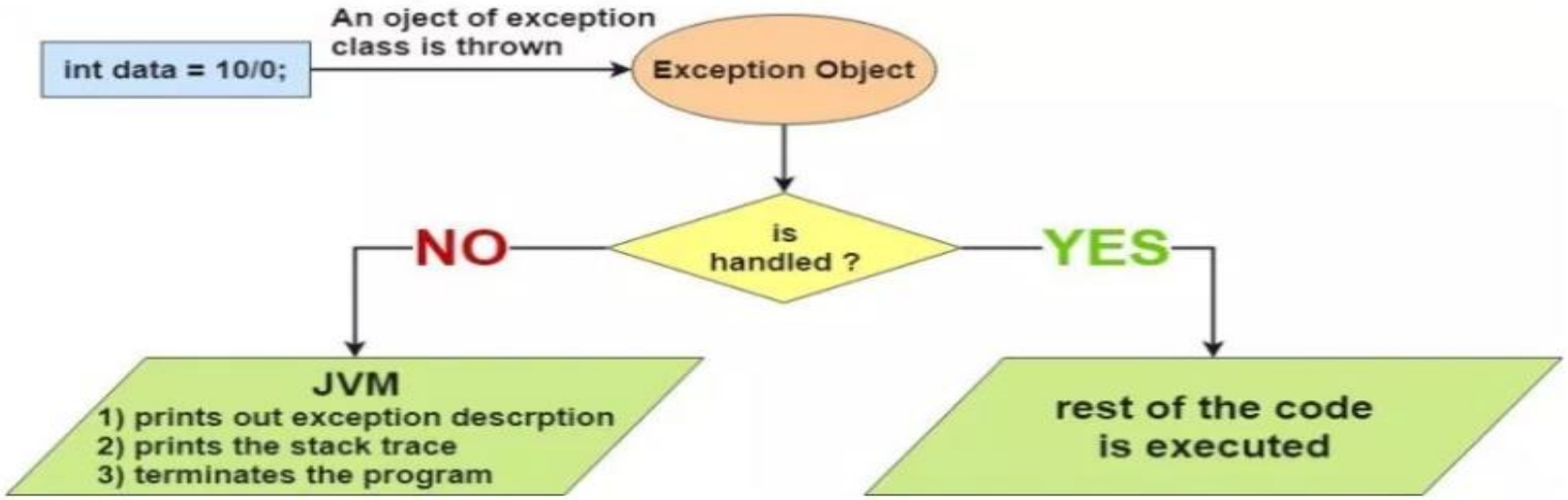
What is an Exception

axess academy

What is Exception and Exception Handling

- **What is an Exception?**
  - An exception is an unwanted or unexpected event, which occurs during the execution of a program i.e. at run time, that disrupts the normal flow of the program's instructions. It is an **object** which is thrown at **runtime**.

- **What is exception handling?**
  - Exception Handling is a mechanism to handle runtime errors such as Arithmetic, ClassNotFound, IO, SQL etc. So it is a way to provide a proper structure when an exception occurs such that the program execution is not affected.
  - Exception Handling is mainly used to handle the checked exceptions. If there occurs any unchecked exception such as NullPointerException, it is programmer's fault that he is not performing check up before the code being used.



axess academy

Common Scenarios of exceptions:

- **Scenario where ArithmeticException occurs :** If we divide any number by zero, there occurs an ArithmeticException.

```
int a=50/0;//ArithmeticException
```

- **Scenario where NullPointerException occurs :** If we have null value in any variable, performing any operation by the variable occurs a NullPointerException.

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

- **Scenario where NumberFormatException occurs: T**he wrong formatting of any value, may occur NumberFormatException.

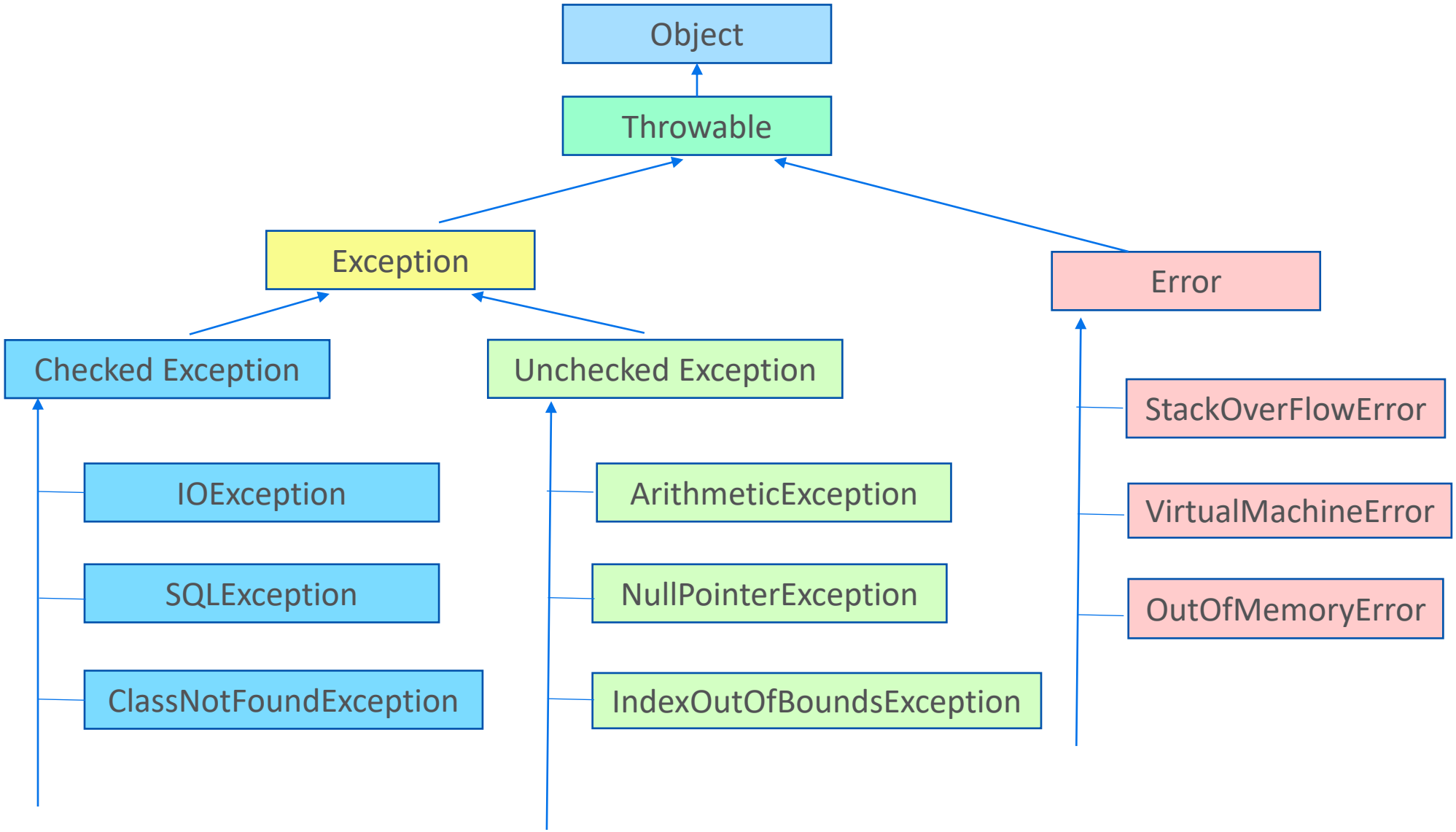
```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

- **Scenario where ArrayIndexOutOfBoundsException occurs:** If you are inserting any value in the wrong index, it would result ArrayIndexOutOfBoundsException.

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

axess academy

Hierarchy of Java Exception classes



Types of Exceptions

- There are basically 3 types of exceptions/abnormal conditions –
  1. **Checked Exceptions:** The classes that extend Throwable class except RuntimeException and Error are known as checked exceptions e.g.IOException, SQLException etc. Checked exceptions are checked at compile-time.

  2. **Unchecked Exceptions:** The classes that extend RuntimeException are known as unchecked exceptions e.g. ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException etc. Unchecked exceptions are not checked at compile-time rather they are checked at runtime.

  3. **Errors:** Error is irrecoverable e.g. OutOfMemoryError, VirtualMachineError, AssertionError etc.

axess academy

Java Exception Handling Keywords

- There are 5 keywords used in java exception handling.
  1. try
  2. catch
  3. finally
  4. throw
  5. throws

axess academy

try, catch and finally Syntax

```
public static void tryCatchFinallyExample (String[] args)
 {
      // array of size 4.
      int[] arr = new int[4];
      try
      {
          int i = arr[4];
          // this statement will never execute
          // as exception is raised by above statement
          System.out.println("Inside try block");
      }
      catch(ArrayIndexOutOfBoundsException ex)
      {
          System.out.println("Exception caught in catch block");
      }
      finally
      {
          System.out.println("finally block executed");
      }
      // rest program will be executed
      System.out.println("Outside try-catch-finally clause");
 }
```

Code Snippet

```
Exception caught in catch block
finally block executed
Outside try-catch-finally clause
```

Output

axess academy

Multi Catch Block

```
public static void MultiCatchBlocks(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...");
      }
```

Code Snippet

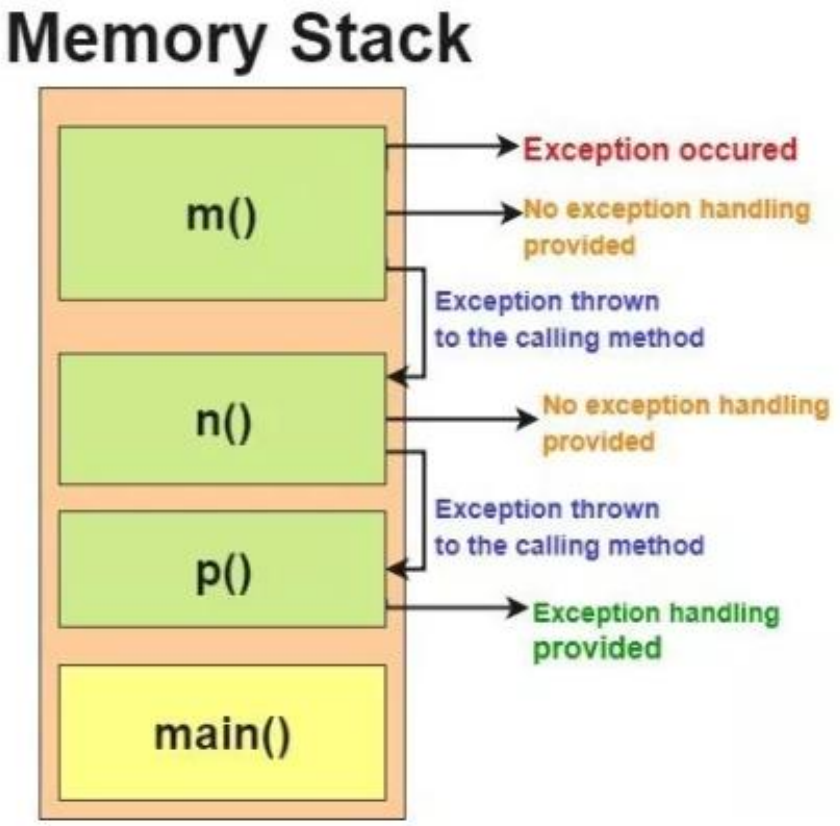
```
Output:task1 completed
rest of the code...
```

Output

axess academy

Exception Propagation

- An exception is first thrown from the top of the stack and if it is not caught, it drops down the call stack to the previous method, If not caught there, the exception again drops down to the previous method, and so on until they are caught or until they reach the very bottom of the call stack. This is called exception propagation.
- It happens only for checked exception.



axess academy

Throw and Throws Keyword

- The Java throw keyword is used to explicitly throw an exception. The throw keyword is mainly used to throw **custom exception (**Checked**).**

- The **Java throws keyword** is used to declare an exception. It gives an information to the programmer that there may occur an exception so it is better for the programmer to provide the exception handling code so that normal flow can be maintained.

```
class ThrowAndThrowsExample{
  void m()throws IOException{
    throw new IOException("device error");//checked exception
  }
  void n()throws IOException{
    m();
  }
  void p(){
   try{
    n();
   }catch(Exception e){System.out.println("exception handled");}
  }
  public static void main(String args[]){
  ThrowAndThrowsExample obj=new ThrowAndThrowsExample();
   obj.p();
   System.out.println("normal flow...");
  }
}
```

axess academy

Throw vs Throws

No.	throw	throws
1.	Java throw keyword is used to explicitly throw an exception.	Java throws keyword is used to declare an exception.
2.	Throw is followed by an instance.	Throws is followed by class.
3.	Throw is used within the method.	Throws is used with the method signature.
4.	You cannot throw multiple exceptions.	You can declare multiple exceptions e.g. public void method()throws IOException,SQLException.

axess academy

Custom Exception

- Create a new class whose name should end with Exception like InvalidAgeException. This is a convention to differentiate an exception class from regular ones.
- Make the class extends one of the exceptions which are subtypes of the java.lang.Exception class.
- Generally, a custom exception class always extends directly from the Exception class so that you will make it a checked exception.
- Create a constructor with a String parameter which is the detail message of the exception. In this constructor, simply call the super constructor and pass the message.

Custom Exception

```
class InvalidAgeException extends Exception{
 InvalidAgeException(String s){
  super(s);
 }
}
```

Code Snippet

```
class TestCustomException{

      static void validate(int age)throws InvalidAgeException{
        if(age<18)
         throw new InvalidAgeException("not valid");
        else
         System.out.println("welcome to vote");
      }

      public static void main(String args[]){
         try{
         validate(13);
         }catch(Exception m){System.out.println("Exception occured: "+m);}

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

Output

```
 Exception occured: InvalidAgeException:not valid
rest of the code..
```

Let us work on use case

axess academy

Bank Customer Management System

- While adding the customer, make sure admin provides valid entries like name can only have alphabets, email should be valid one , contact no should have 10 digits and account type can be either Savings or current.

```
Please enter customer details :
Enter name :
32d328
Name can only have alphabets
```

- If any one of the condition fails, the customer should not be added in the array and display the appropriate error message using custom exceptions.

- Only for valid entries, it should be added in array.

- While taking the input from the user to search a customer using customer id, make sure it has to be an integer else display the appropriate error message using custom exceptions.

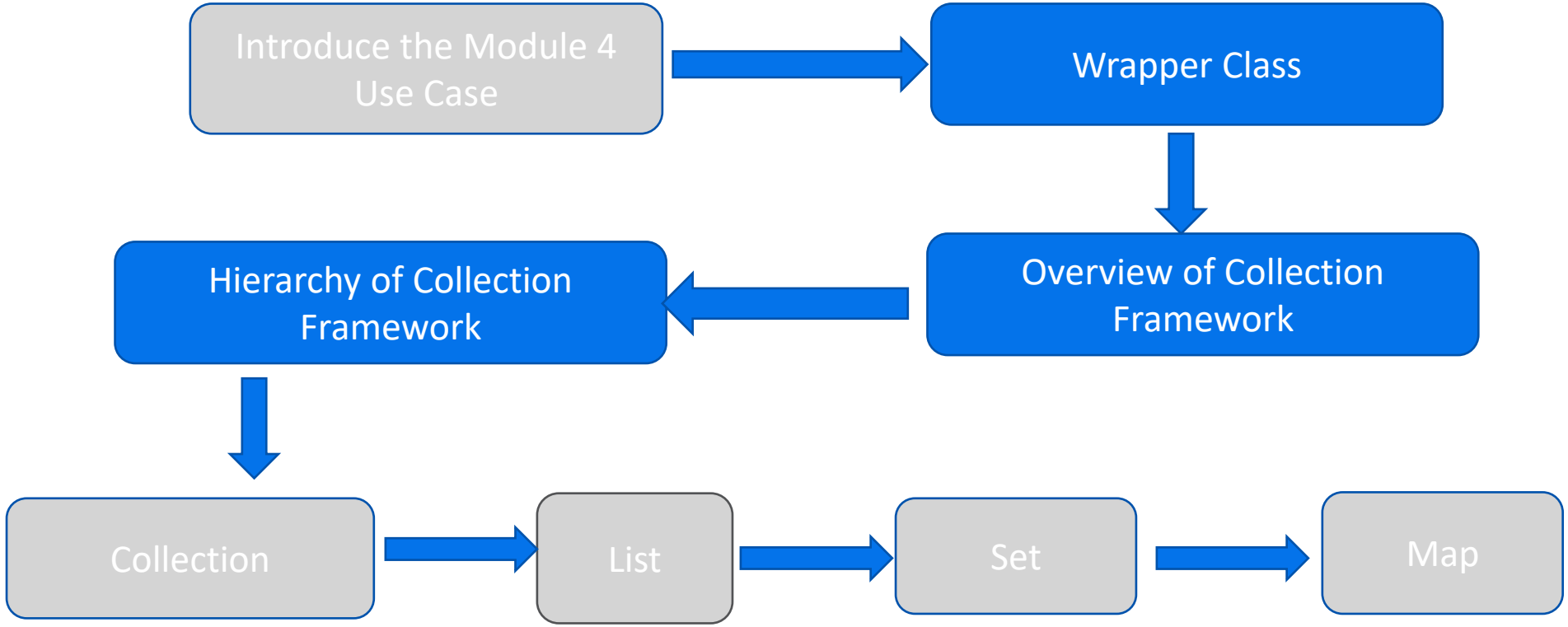
axess academy

Module 4

Collection Framework

Module 4 : Collection Framework

axess academy



Introducing the use case

axess academy

Bank Customer Management System

- Convert array into ArrayList and make the corresponding code changes.
- If user enters 4 , delete the matched customer details.

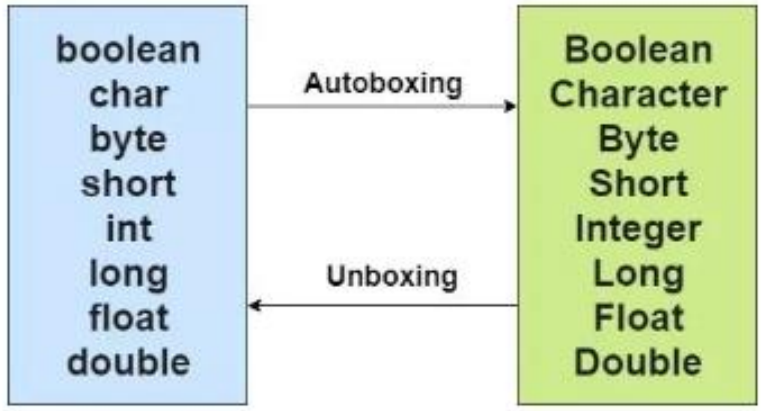
```
4
Please enter customer id to be deleted
1014
Deleted customer with id =1014
```

Wrapper Class

Wrapper Class

- **Wrapper classes in java** provides the mechanism *to convert primitive into object and object into primitive*.

- Since J2SE 5.0 autoboxing and unboxing feature converts primitive into object and object into primitive automatically. The automatic conversion of primitive into object is known as autoboxing and vice-versa unboxing.




```
// autoboxing  - primitive object to Character conversion
 Character ch = 'a';
  // unboxing - Character object to primitive conversion
  char a = ch;
  System.out.println("Character ch: "+ch);
  System.out.println("char a: "+a);
```

axess academy

Collection Framework

axess academy

Overview of Collection Framework

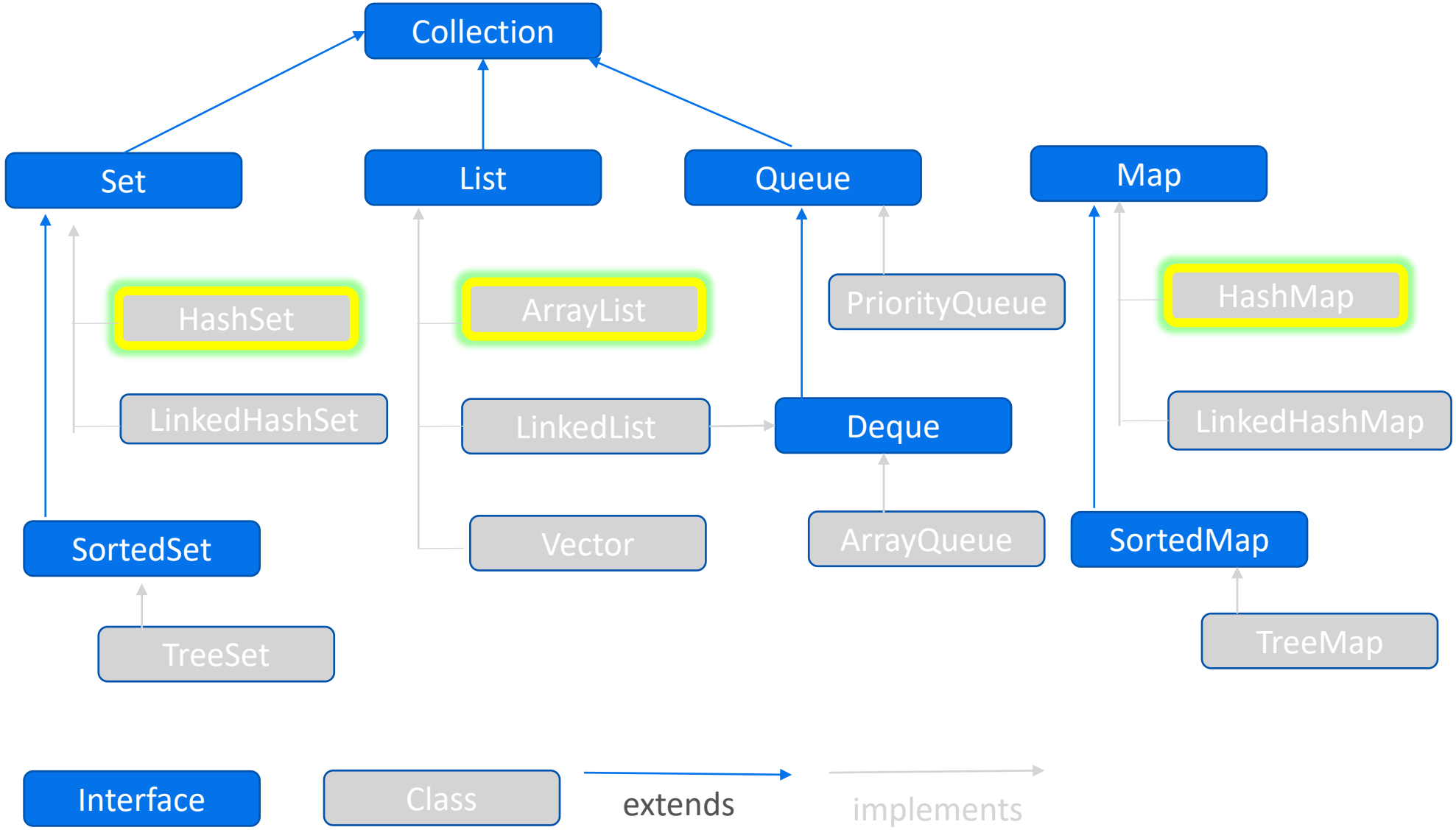
- The **Java collections framework** is a set of classes and interfaces that implement commonly reusable collection data structures.

- Although referred to as a framework, it works in a manner of a library. The collections framework provides both interfaces that define various collections and classes that implement them.

- Java Collection framework provides many interfaces (Set, List, Queue, Deque) and classes (ArrayList, Vector, LinkedList, PriorityQueue, HashSet, LinkedHashSet, TreeSet).

axess academy

Collection Framework Hierarchy



Collection

- The Collection interface is the foundation upon which the collections framework is built. It declares the core methods that all collections will have.

- Some of the methods of Collection interface are Boolean add (Object obj), Boolean addAll (Collection c), void clear(), etc. which are implemented by all the subclasses of Collection interface.

Method	Description	Code
boolean add(Object obj)	Adds obj to the invoking collection. Returns true if obj was added to the collection. Returns false if the collection does not allow duplicates.	Collection collection = new HashSet(); collection.add(“value”);
boolean contains(Object obj)	Returns true if obj is an element of the invoking collection. Otherwise, returns false.	collection.contains(“test”);=>false
boolean remove(Object obj)	Removes one instance of obj from the invoking collection. Returns true if the element was removed. Otherwise, returns false.	collection.remove(“value”); => value
int size( )	Returns the number of elements held in the invoking collection.	collection.size() => 0
boolean isEmpty( )	Returns true if the invoking collection is empty. Otherwise, returns false.	Collection.isEmpty() => true
Iterator iterator( )	Returns an iterator for the invoking collection.	Collection.iterator()

axess academy

List Interface

- A List is an ordered Collection (sometimes called a *sequence*). Lists may contain duplicate elements.

- List interface is implemented by the classes ArrayList, LinkedList, Vector, and Stack.
  - **ArrayList**: Implemented as a single linked list.
  - **LinkedList:** Implemented as a double linked list.
  - **Vector**: Almost identical to ArrayList, and the difference is that Vector is synchronized. Because of this, it has an overhead than ArrayList.

```
//Creating arraylist
List<String> list=new ArrayList<String>();
//List<String> list=new LinkedList<String>();
list.add("John");//Adding object in arraylist
list.add("Jack");
list.add("Ben");
list.add("Bella");
list.add("Ben"); // Adding Duplicate elements

//getting elements at an index
System.out.println("Element at an index 2 is "+list.get(2));
//Traversing list through Iterator
Iterator<String> itr=list.iterator();
System.out.println("Elements in the list");
while(itr.hasNext()){
//as its ordered collection so
//elements will be displayed in insertion order)
System.out.println(itr.next());
}
```

Code Snippet

Output

```
Element at an index 2 is Ben
Elements in the list
John
Jack
Ben
Bella
Ben
```

axess academy

Set Interface

- A Set is an unordered Collection that cannot contain duplicate elements.
- Set interface is implemented by the classes HashSet, LinkedHashSet and TreeSet.
  - **HashSet**: Unordered Collection with no duplicates
  - **LinkedHashSet**: Ordered Collection with no duplicates
  - **TreeSet:** Sorted Collection with no duplicates

```
//Creating Set
Set<String> set=new HashSet<String>();
//Set<String> set=new LinkedHashSet<String>();
//Set<String> set=new TreeSet<String>();
set.add("John");//Adding object in Set
set.add("Jack");
set.add("Ben");
set.add("Bella");
set.add("Ben"); // Adding Duplicate elements

//Traversing Set through Iterator
Iterator<String> itr=set.iterator();
System.out.println("Elements in the Set");
while(itr.hasNext()){
//as its unordered collection so
//elements may or may not be displayed in insertion order)
System.out.println(itr.next());
}
```

Code Snippet

```
Elements in the Set
John
Ben
Jack
Bella
```

HashSet Output

```
Elements in the Set
John
Jack
Ben
Bella
```

LinkedHashSet Output

```
Elements in the Set
Bella
Ben
Jack
John
```

TreeSet Output

axess academy

Map Interface

- A Map is an object which maps keys to values. A map cannot contain duplicate keys and each key can map to at most one value
- Map interface is implemented by the classes HashMap, LinkedHashMap and TreeMap.
  - **HashMap**: Unordered with no duplicate keys.
  - **LinkedHashMap**: Ordered with no duplicate keys.
  - **TreeMap**: Sorted keys with no duplicate keys

```
//Creating Map
Map<Integer,String> map=new HashMap<Integer,String>();
//Map<Integer,String> map=new LinkedHashMap<Integer,String>();
//Map<Integer,String> map=new TreeMap<Integer,String>();
map.put(101,"John");//Adding object in Map
map.put(105,"Jack");
map.put(103,"Ben");
map.put(104,"Bella");
map.put(102,"Ben"); // Adding Duplicate keys

//Returns the set of keys
Set<Integer> set= map.keySet();

Iterator<Integer> itr=set.iterator();
System.out.println("Elements in the map");
while(itr.hasNext()){
//as its unordered collection so
//elements may or may not be displayed in insertion order)
Integer key= (Integer)itr.next();
System.out.println("key ="+key+" value ="+map.get(key));
}
```

Code Snippet

```
Elements in the map
key =101 value =John
key =102 value =Ben
key =103 value =Ben
key =104 value =Bella
key =105 value =Jack
```

HashMap Output

```
Elements in the map
key =101 value =John
key =105 value =Jack
key =103 value =Ben
key =104 value =Bella
key =102 value =Ben
```

LinkedHashMap Output

```
Elements in the map
key =101 value =John
key =102 value =Ben
key =103 value =Ben
key =104 value =Bella
key =105 value =Jack
```

TreeMap Output

Let us work on use case

axess academy

Bank Customer Management System

- Convert array into ArrayList and make the corresponding code changes.
- If user enters 4 , delete the matched customer details.

```
4
Please enter customer id to be deleted
1014
Deleted customer with id =1014
```

axess academy