Java
Object Oriented Programming
Structure

Content


Object Oriented
Programming

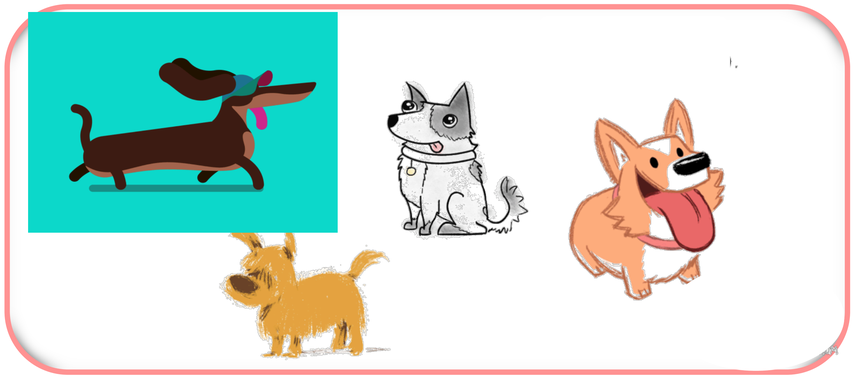
Object Oriented Programming (OOPs)

Object-Oriented Programming is a methodology or paradigm to design a program using Classes and Objects

Object Oriented Programming (OOPs)

Real world entities that has their own properties and behaviours

Class Dog



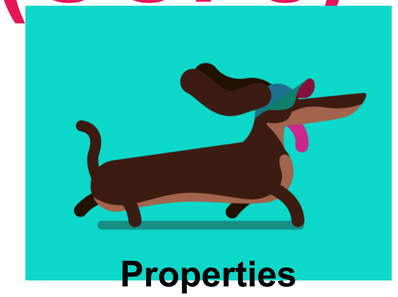
Blueprint from which an objects properties and behaviours are decided

Properties	Behaviour
breed	eat()
size	sleep()
age	run()
color	bark()

Object Oriented Programming(OOPs)

For different values of data members (breed size, age, and color) in Java class, you will get a different dog object

Object Oriented Programming (OOPs)



Properties

breed

size

age

color

Behaviour

eat()

sleep()

run()

bark()

Properties

breed

size

age

color

Behaviour

eat()

sleep()

run()

bark()

Properties

breed

size

age

color

Behaviour

eat()

sleep()

run()

bark()

Properties

breed

size

age

color

Behaviour

eat()

sleep()

run()

bark()

Java Objects

A real-world entity that has state and behaviour is known as an object

State 01

It is the data (value) of an object

02 Behavior

It is the functionality) of an object

03 Identity

The object identity is typically implemented via a unique ID that is used internally by the JVM

CHARACTERISTICS

Java Classes

A class is a **blueprint of object** having properties

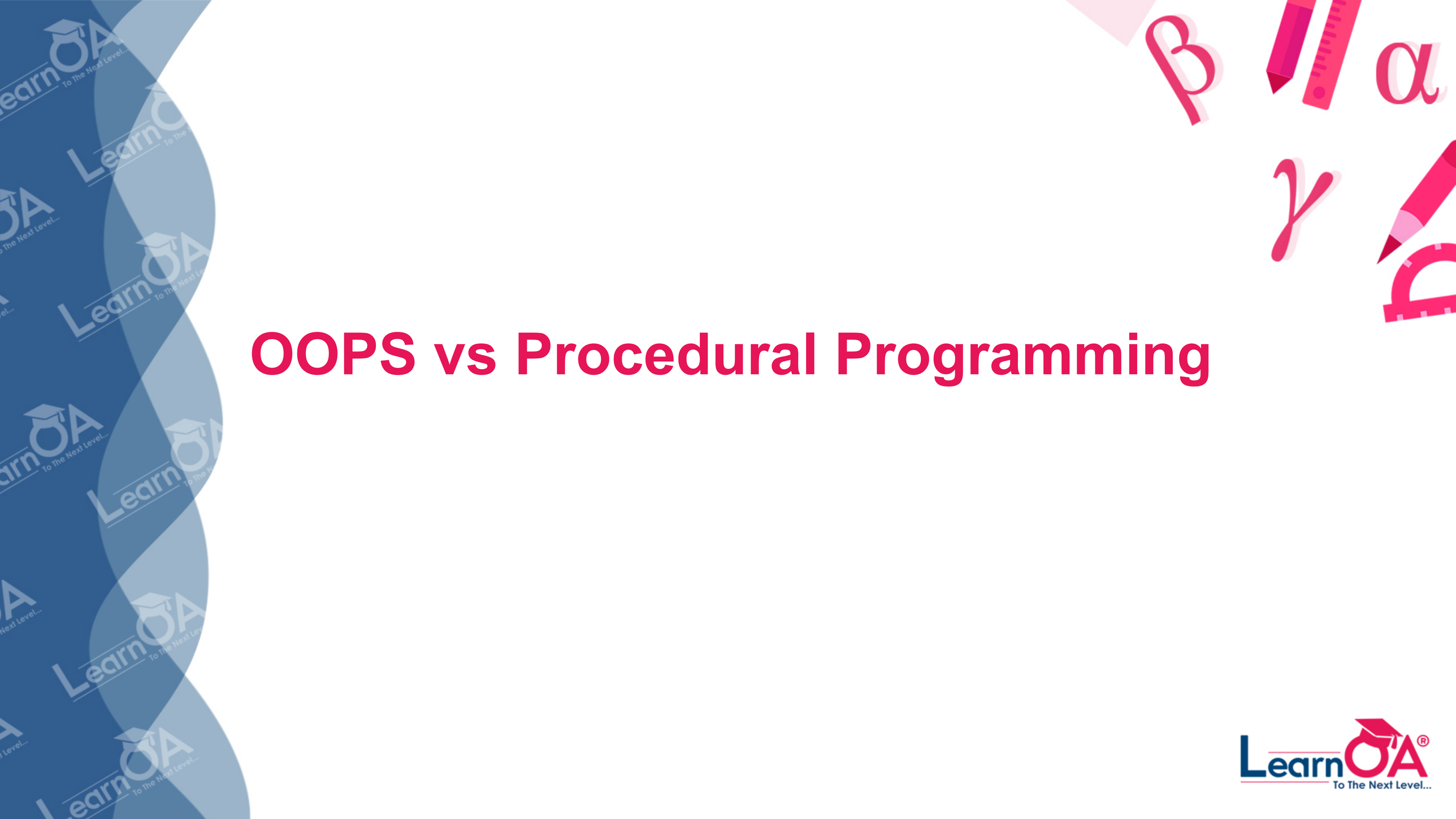
```
class <class_name>{
    field;
    method;
}
```

Java Classes (Contd.)

A class may contain

```
class <class_name>{
    field;
    method;
}
```

- Fields
- Methods
- Constructors
- Blocks
- Nested Class & Interface


OOPS vs Procedural Programming

Object Oriented Programming

- ✓ Bottom Up approach
- ✓ Divided into objects
- ✓ Has ***Access Modifiers***
- ✓ Objects can move & communicate with each other through member functions
- ✓ More secure
- ✓ Supports ***overloading***

Procedural Programming

- ✓ Top Down Approach
- ✓ Divided into functions
- ✓ Doesn’t have Access Modifiers
- ✓ Data can move freely from function to function in the system
- ✓ Less Secure
- ✓ Do not support overloading

Java Methods

Java Methods

A method is a set of code that is grouped together to perform a specific operation

A method must be written inside a class

Each method has its own signature

Java provides two types of methods

- Pre Defined or Standard Library Methods
- User Defined Methods

Java User Defined Methods

To use a method, you need to perform two steps:

- Method Initialization
- Method Invocation

Java Methods

Method Initialization

```
modifier returnType nameOfMethod (Parameter List)
{
    // method body
}
```

- ✓ A method can be parameterized or non-parameterized
- ✓ Method definition consists of a method header and a method body
- ✓ You can ***Overload Method*** i.e. Provide same name to more than one method but their data type or parameter list must be different

Java Methods

Method Invocation

```
methodName()

methodName(parameter1, parameter2...)
```

- ✓ To use a method it needs to be invoked or called
- ✓ When a program invokes a method, the program control gets transferred to the called method
- ✓ A method can be called in two ways:
  - Call by Value
  - Call by Reference


Static Vs Non-Static

Static vs Non Static

Non-static variable	Static variable
• Non-static variable also known as instance variable while because memory is allocated whenever is created. • Non-static variable are specific to an object. • Non-static variable can access with object reference. • **Syntax**	• Memory is allocated at the time of loading of class so that these are also known as class variables. • Static variable are common for every object that mean these memory location can be shareable by every object reference or same class. • static variable can access with class reference. • **Syntax**
Obj_ref.variable_name	class_name.variable_name

Static vs Non Static

Non-static Method	Static Method
• These method never be preceded by static keyword **Example:**	• These method always preceded by static keyword **Example:**

`void fun()`
`{`
`.....`
`.....`
`}`
	`static void fun()`
	`{`
	`.....`
	`.....`
	`}`

• Memory is allocated multiple time whenever method is calling.	• Memory is allocated only once at the time of class loading.

Constructors

Constructors

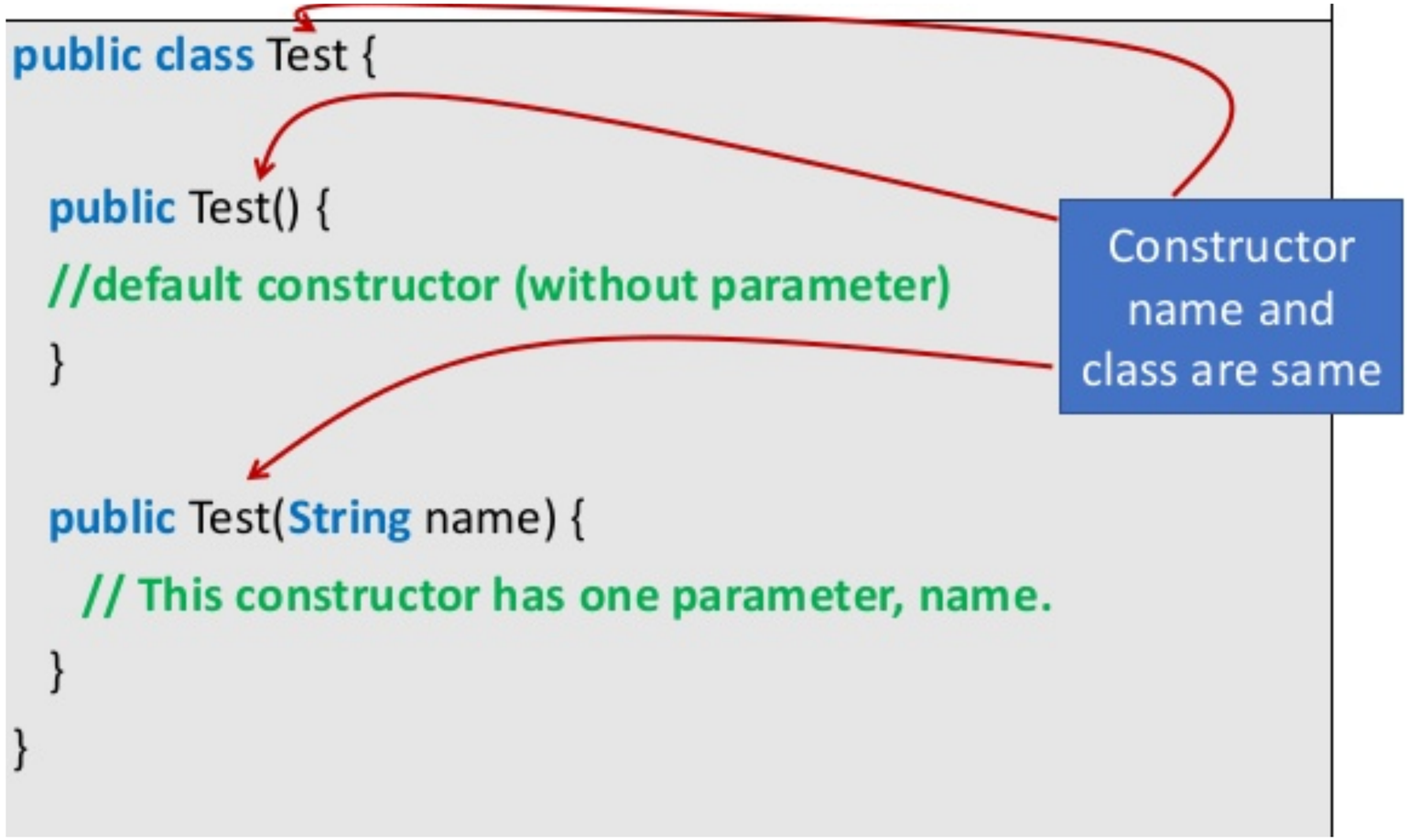
- When discussing about classes, one of the most important sub topic would be constructors. Every class has a constructor. If we do not explicitly write a constructor for a class, the Java compiler builds a default constructor for that class.

- Each time a new object is created, at least one constructor will be invoked. The main rule of constructors is that they should have the same name as the class. A class can have more than one constructor.

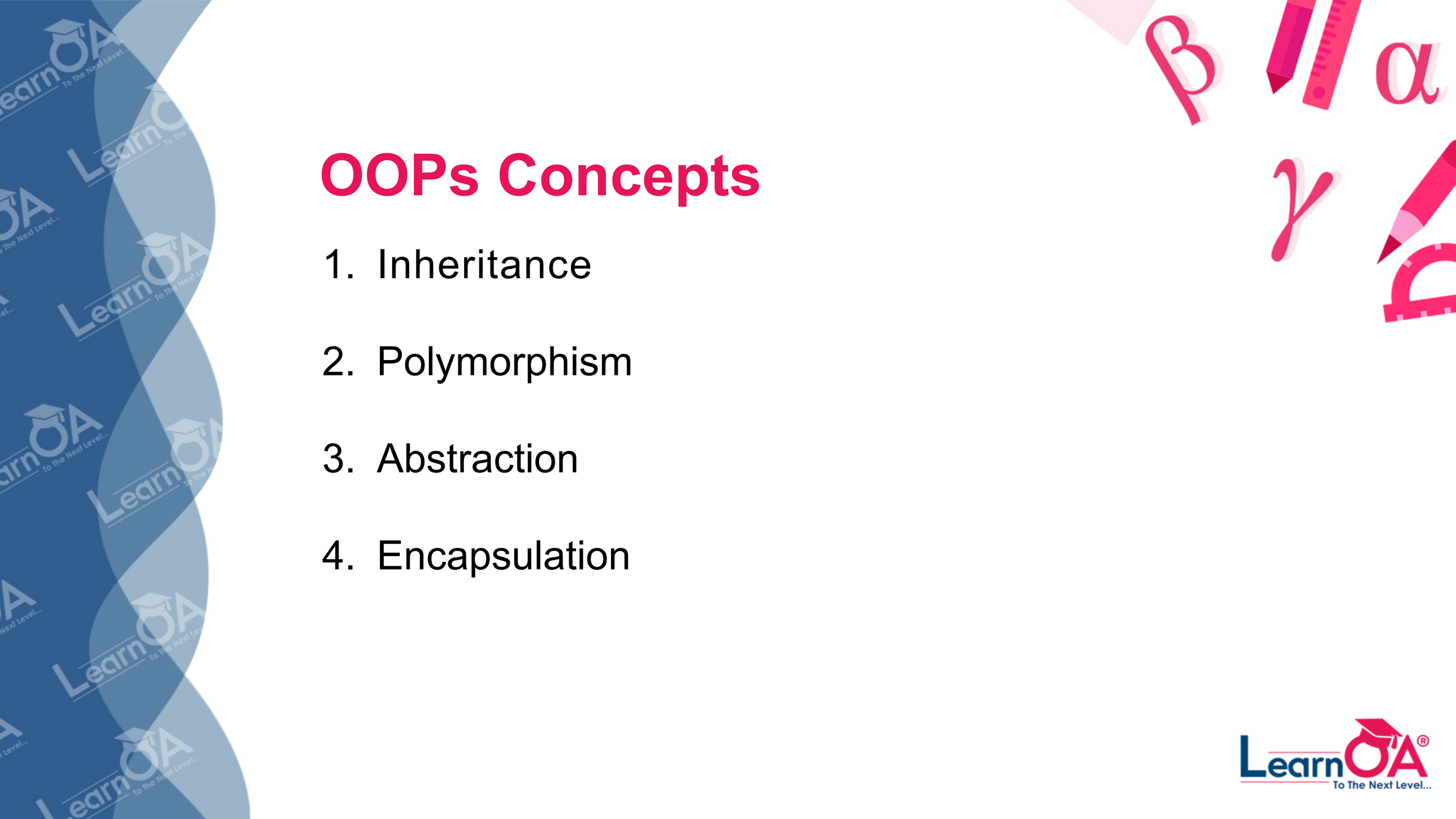
Constructors (Contd.)

- Constructor is a class member function with same name as the class.
- The main job of constructor is to allocate memory for class objects.
- Constructor must be in pubic area of the class
- Constructor is automatically called when object is created.

Constructors (Contd.)



OOPs Concepts

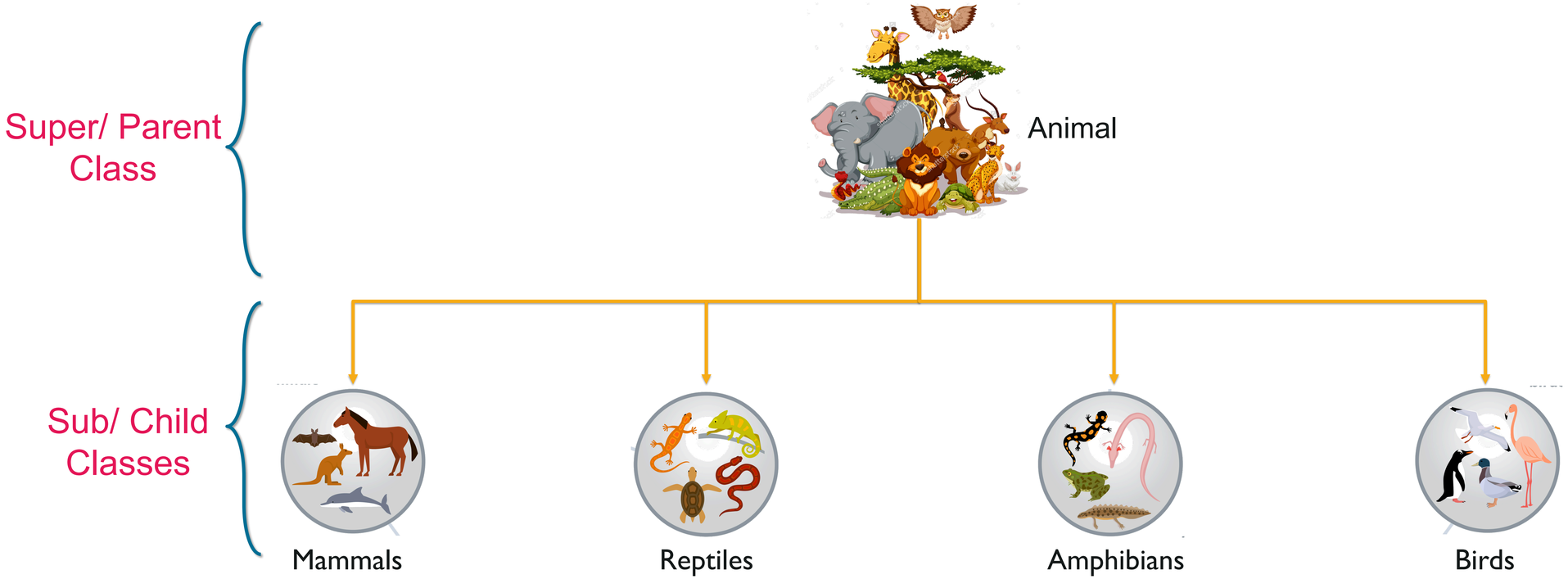


OOPs Concepts

1. Inheritance
2. Polymorphism
3. Abstraction
4. Encapsulation

Inheritance

- Inheritance is the property of an object to acquire all the properties and behavior of its parent object
- Inheritance represents the IS-A relationship which is also known as a parent-child relationship



Inheritance

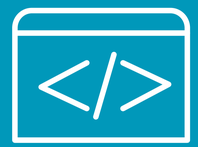
Syntax

```
class Subclass extends Superclass
{
    //methods and fields
}
```

Advantages

Code Reusability

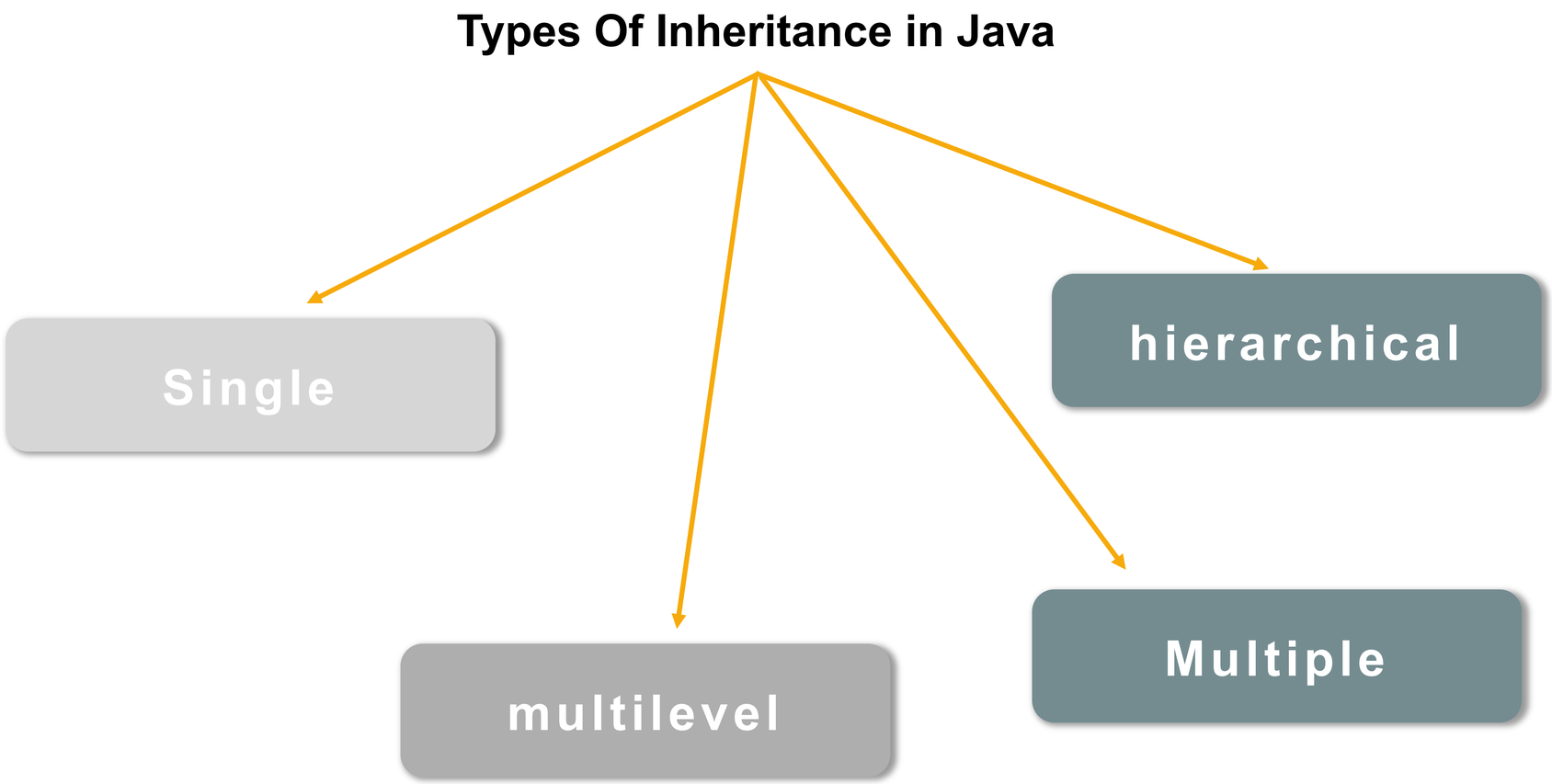
Extensibility



Overriding

Data Hiding

Inheritance

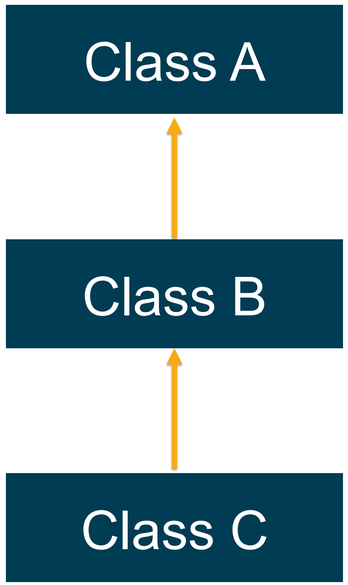


Inheritance - Single

- Single level inheritance enables a derived class to inherit properties and behaviour from a single parent class

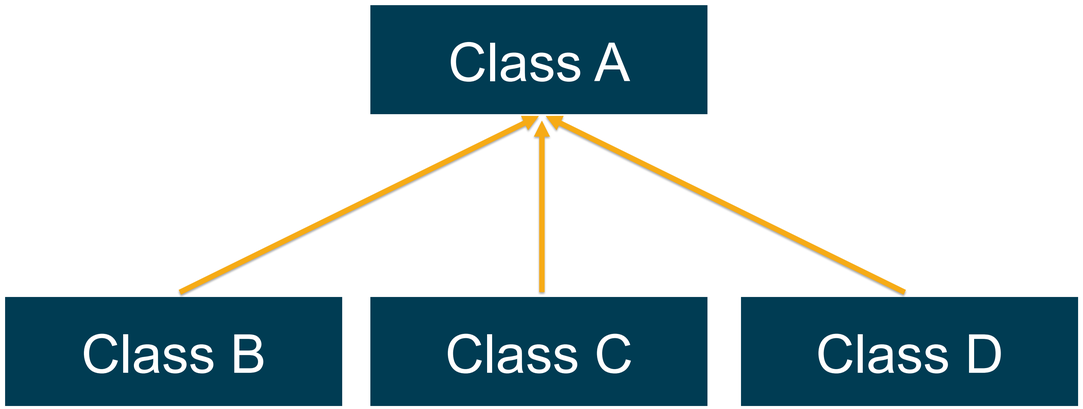
Inheritance - Multilevel

- Multi level inheritance enables a derived class to inherit properties and behaviour from a parent class which is also derived from another class



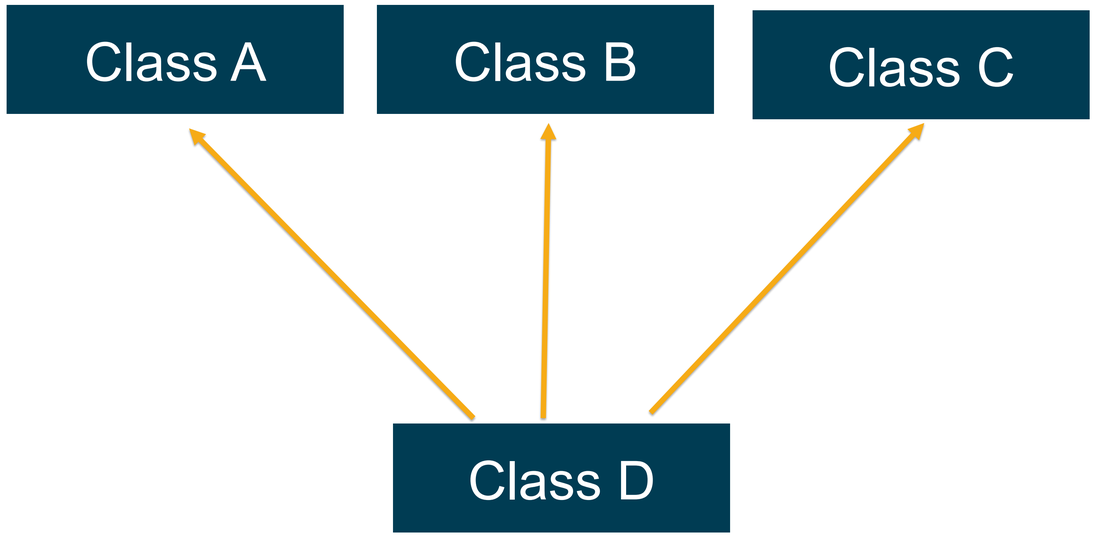
Inheritance - Hierarchical

- Hierarchical level inheritance enables more than one derived class to inherit properties and behaviour from a parent class



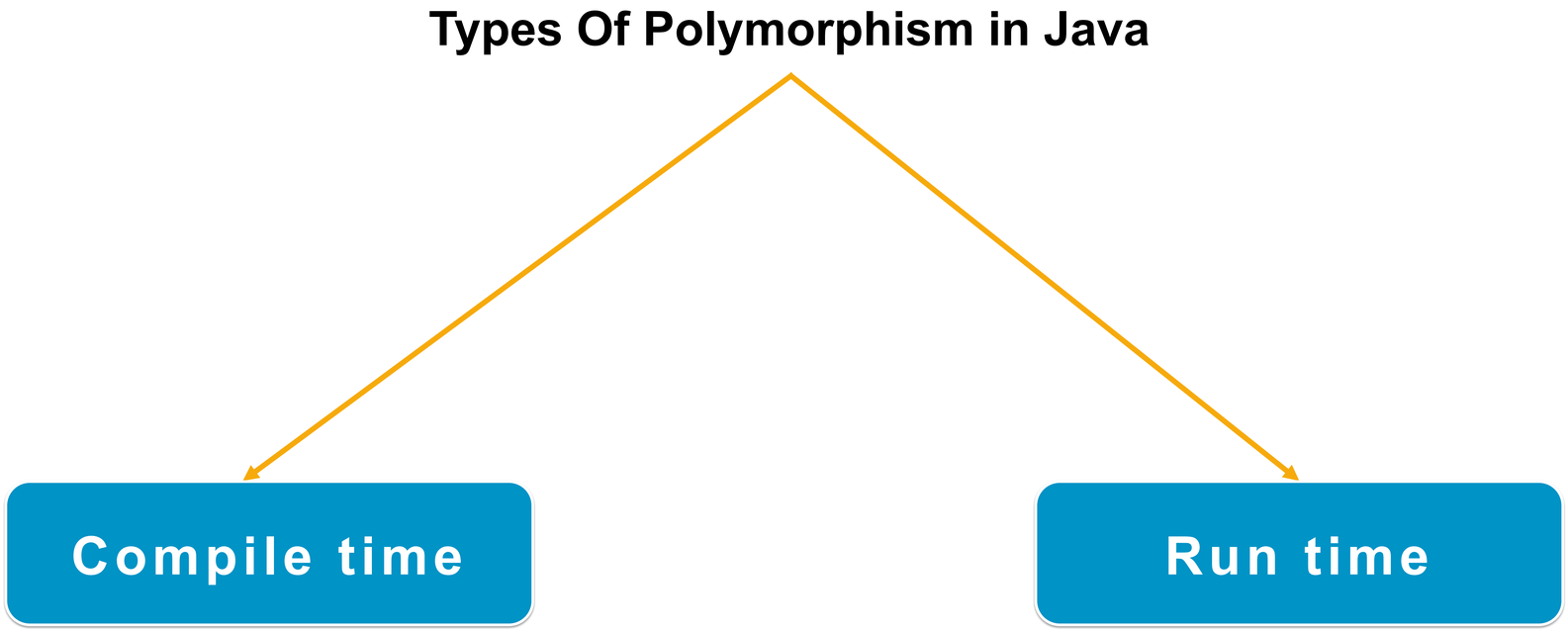
Inheritance - Multiple

- When We inherit from multiple classes i.e. more than 1 class.
- Multiple Inheritance is not supported by Java.



Polymorphism

- Polymorphism is the property of an object which allows it to take multiple forms



Polymorphism – Compile Time

- Compile Time Polymorphism or Static Polymorphism is resolved during compiler time
- **Overloading** is an example of compile time polymorphism

Rules For Overloading

1. Overloaded methods must have different argument list
2. It can have different return types if argument list is different
3. It can throw different exceptions
4. It can have different access modifiers

Polymorphism – Run Time

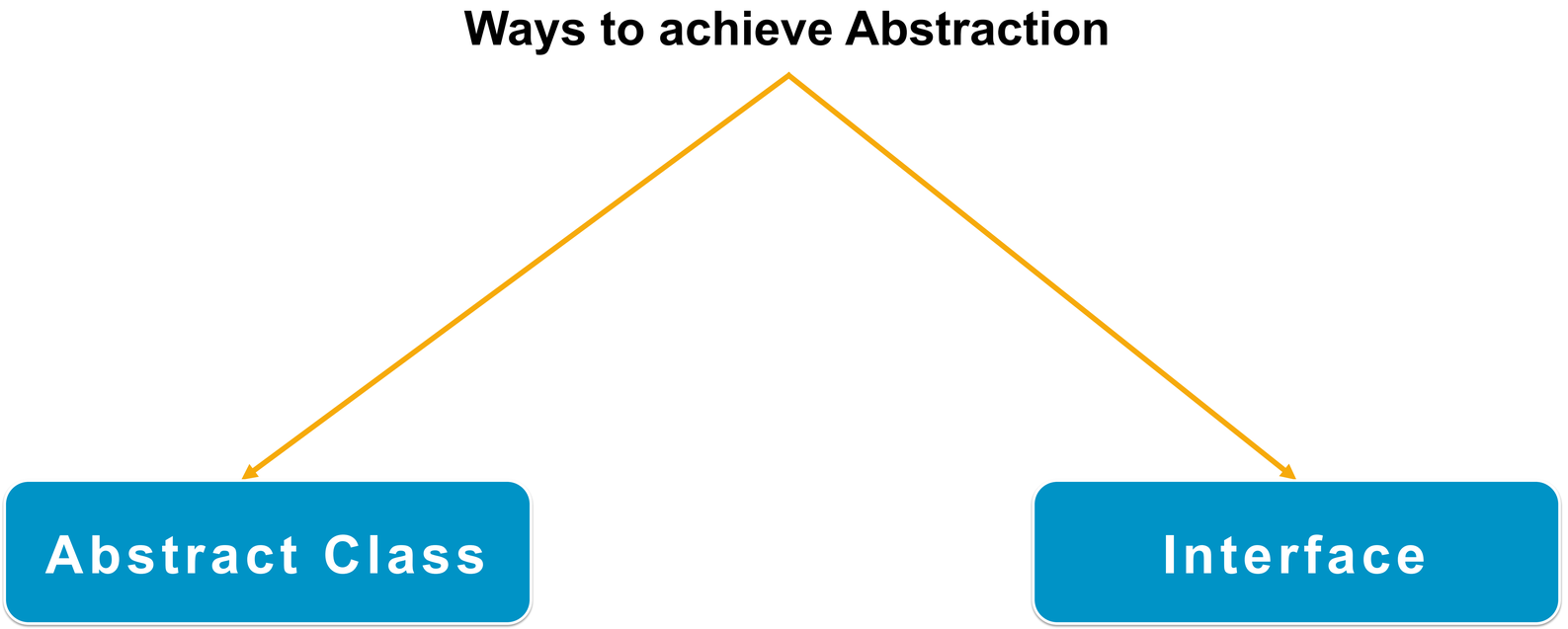
- Run Time Polymorphism or Dynamic Polymorphism is resolved during run time
- Method Overriding is an example of run time polymorphism
- An overridden method is called through the reference variable of a superclass

Rules For Overloading

1. Overriding method argument list must match the overridden method
2. The return type must be the same or subtype of overridden method
3. Access level cannot be more restrictive than overridden method

Abstraction

- Abstraction is the methodology of hiding the implementation details from the user and only providing the functionality to them



Abstraction – Abstract Class

- An abstract class is a template definition to methods and variables of a class that contains one or more abstracted methods

It can provide from 0 to 100% of abstraction

Must be declared with an abstract keyword

Can have abstract and non-abstract methods

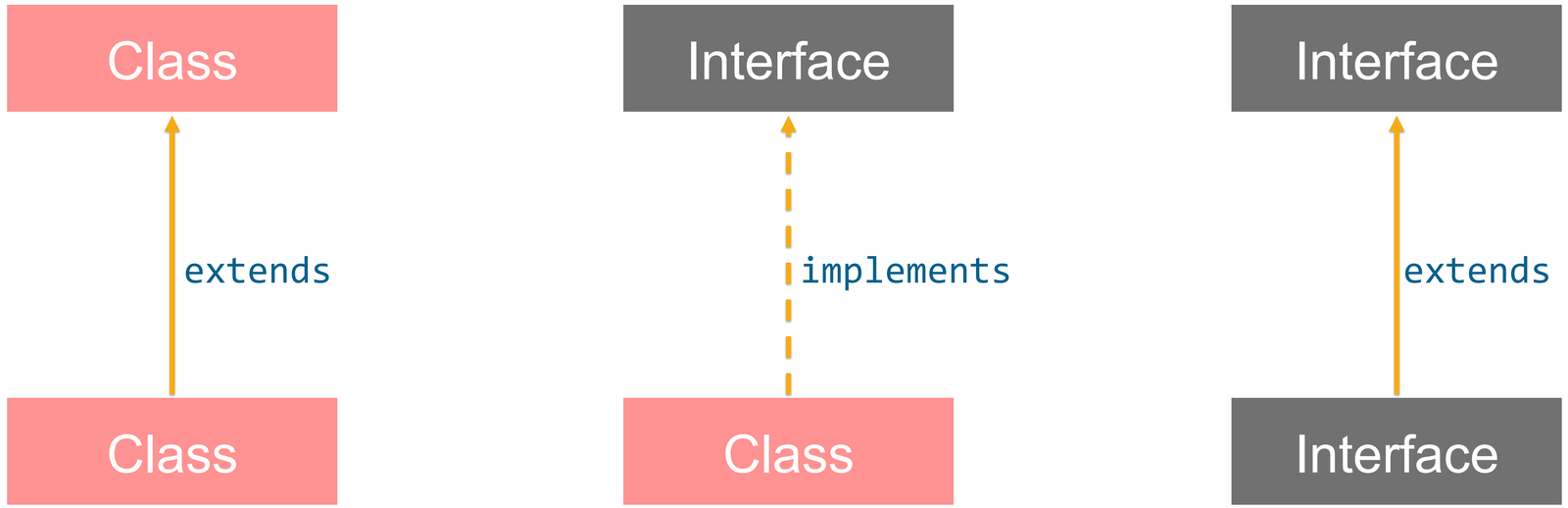
Cannot be instantiated

Can have constructors and static methods

Can have final methods

Abstraction - Interface

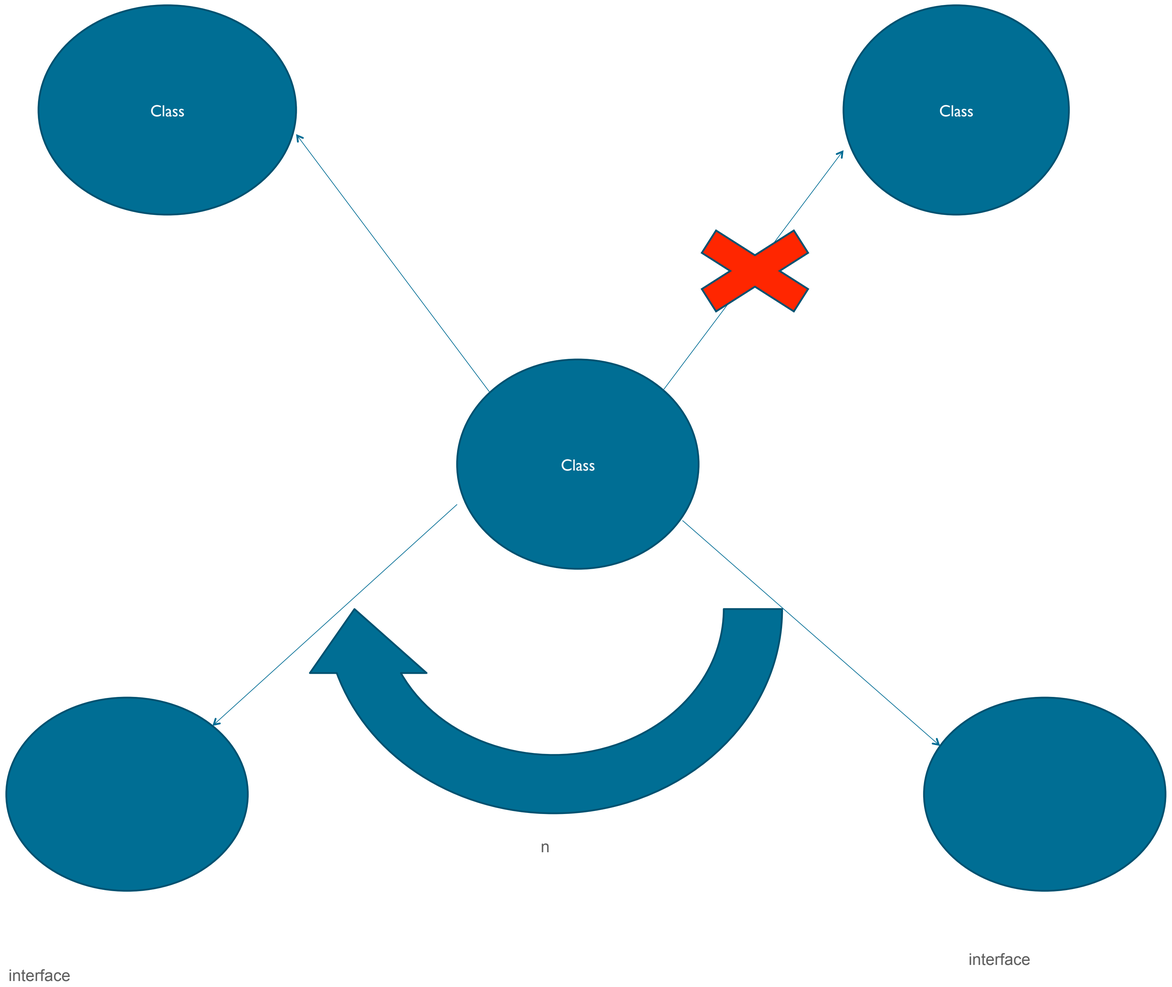
- An interface in java is a blueprint of a class which contains static constants and abstract methods
- It Enables Multiple Inheritance and helps in achieving loose coupling
- **It provides 100% of abstraction**



Why are “Interfaces” used?

Interfaces are used to implement the expected behavior of a system/data types.

Java does not support multiple inheritance, hence interfaces are implemented.


Class
Class
Class
n
interface
interface

Interface

An interface has a set of method declarations.

It does not have the method body but only method declaration.

To use an interface in a class, “implements” keyword is used.

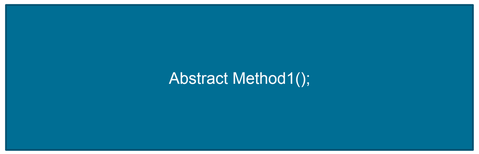
In case you don’t override all the methods in the class the class has to be defined as abstract.

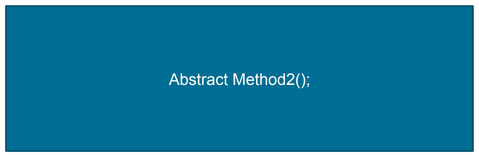
An interface is same as class except class can be instantiated but interface cannot be.

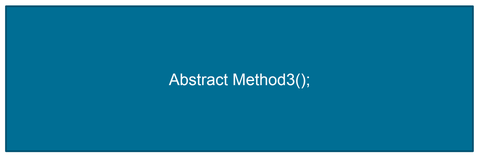

INTERFACE

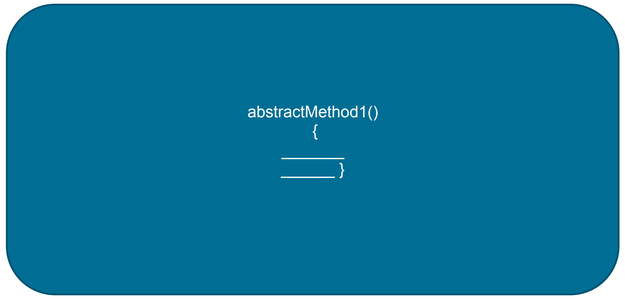
Implements

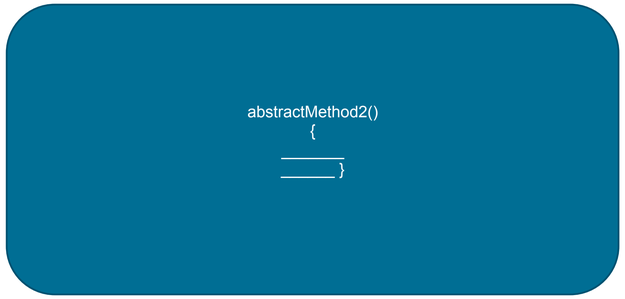

Class

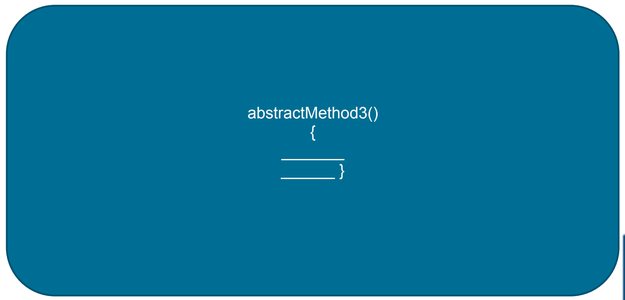

Abstract Method1();

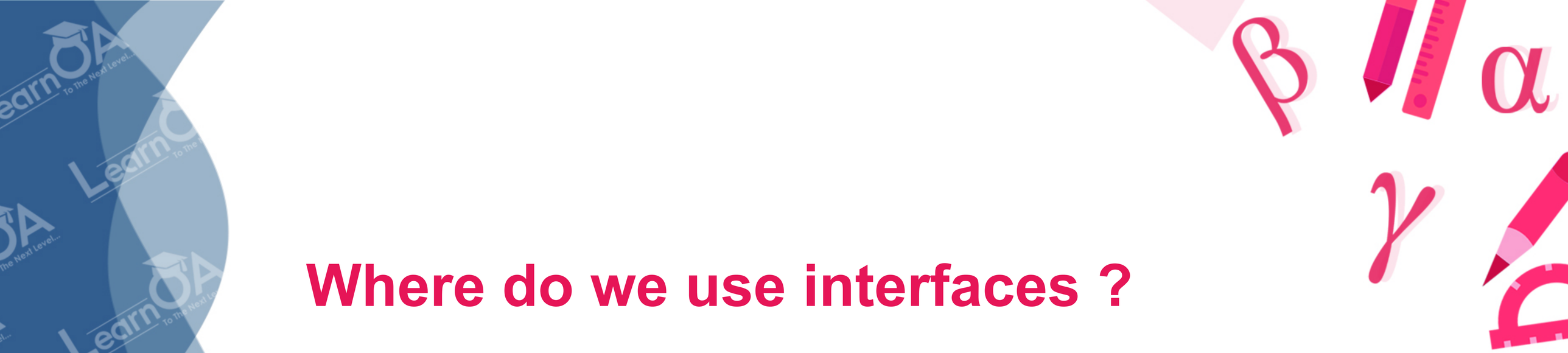

Abstract Method2();


Abstract Method3();


abstractMethod1()
{
_____
_____ }


abstractMethod2()
{
_____
_____ }


abstractMethod3()
{
_____
_____ }



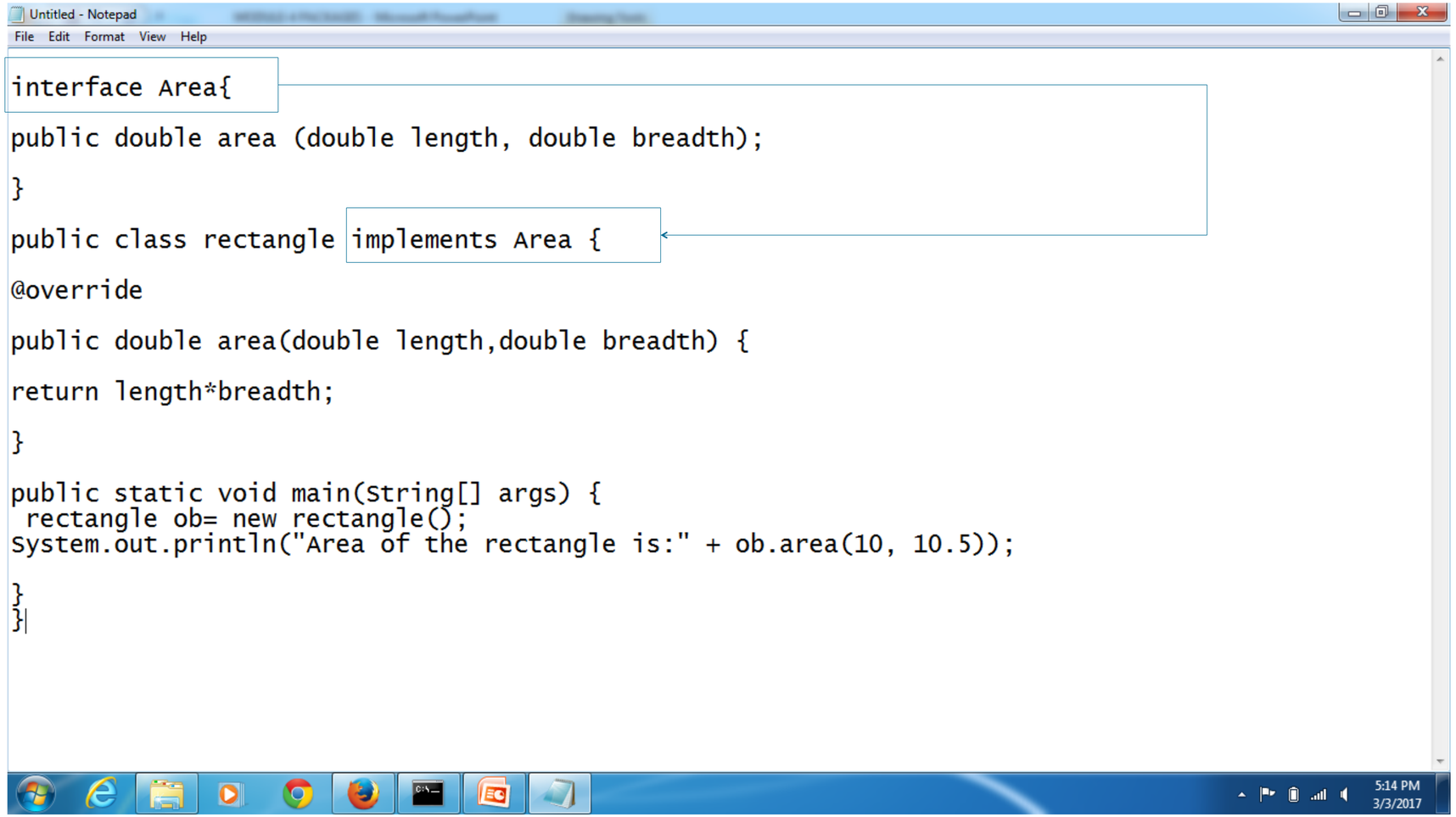
Where do we use interfaces ?

Any system which needs the expected functionality, requires interfaces.

For example , 2 basic functions of a bank is deposit and withdraw .

These two functions can be part of interfaces for banking applications.

A Program on Interface

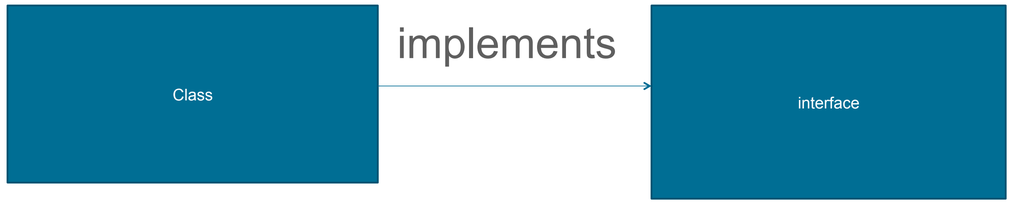


Interfaces (Contd.)

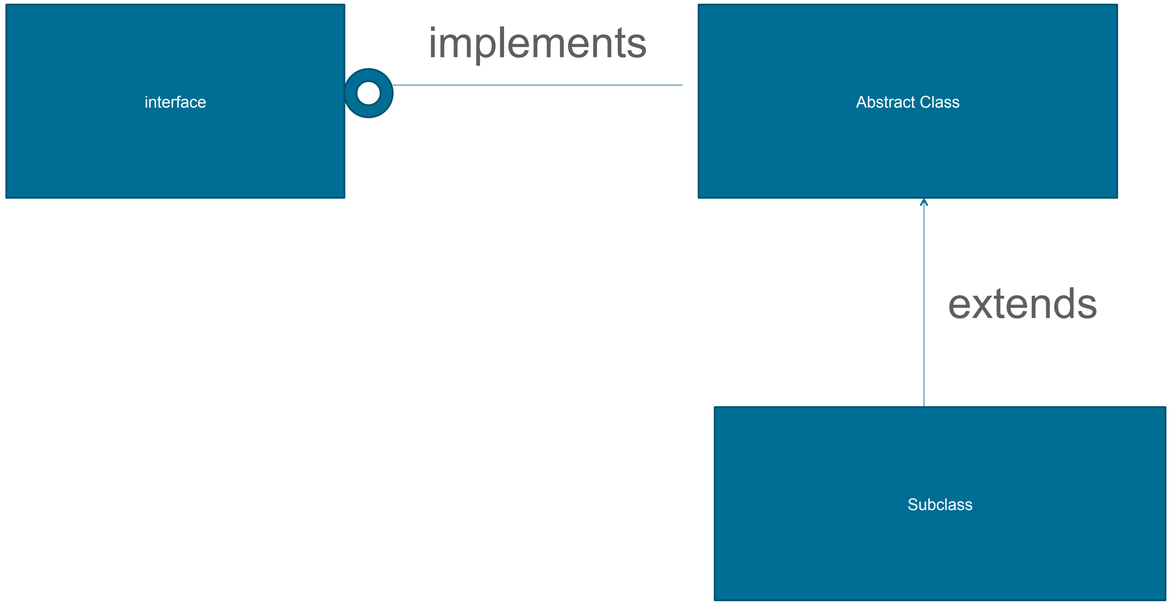
Attributes can be defined in interfaces.

These attributes can be used in the implemented class.

The attribute values can’t be changed in the class as it acts like final variables.

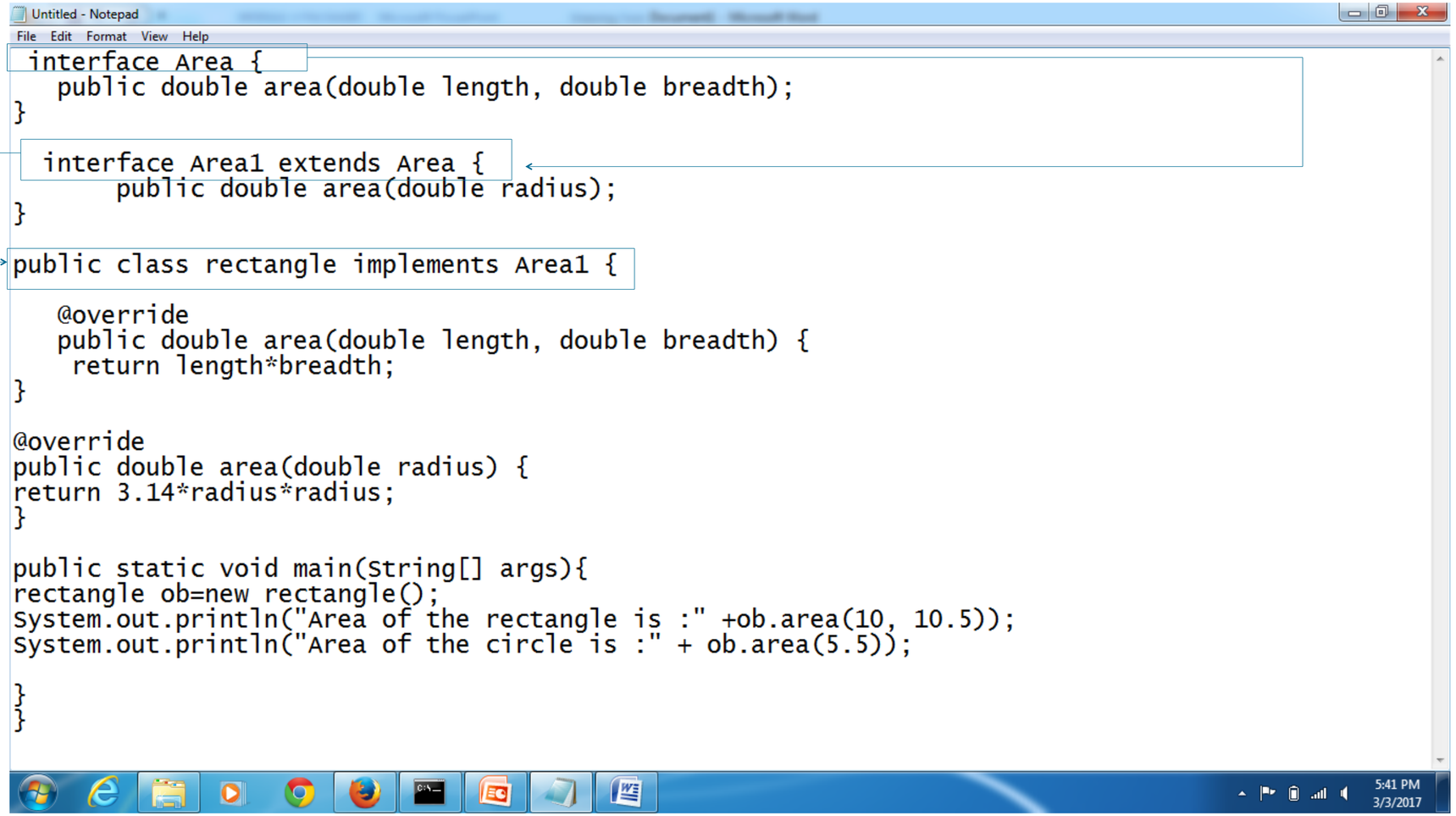


Class implementing an interface



class extending an Abstract class which implements an interface.

Interfaces can be Extended



Interfaces can be Extended (Contd.)

Like the classes can be extended, even interfaces can be extended .

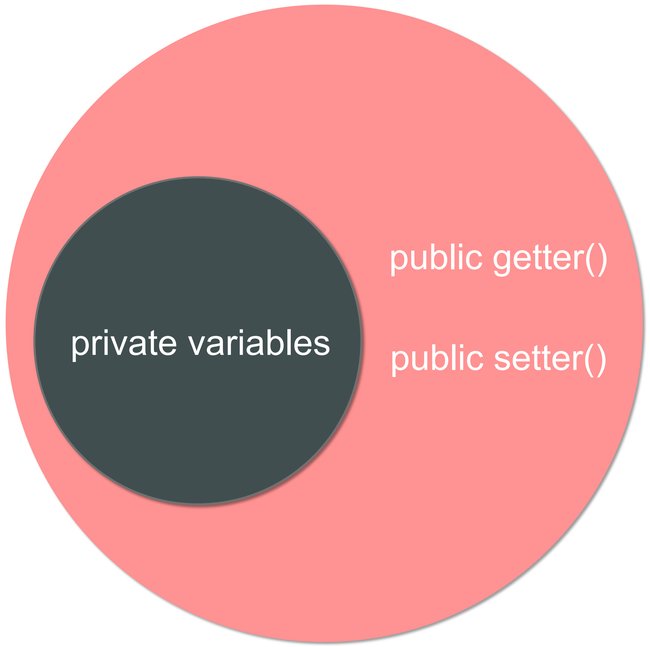
In the above program, there are two interfaces, interface 1 and interface 2.

Interfaces 2 is extended from interface1. If a class is implementing interface2 then all the methods of interface2 and 1 should be implemented , as interface2 is extended from interface1.

Encapsulation

- Encapsulation is the mechanism of wrapping up of data and code acting on the methods together as a single unit

- It is achieved by declaring the variables of a class as private and then providing the public setter and getter methods to modify and view the variables values



Anonymous Objects

Anonymous Objects

- Anonymous simply means nameless. An object that have no reference is known as anonymous object.
- If you have to use an object only once, anonymous object is a good approach.
- The anonymous object is created and dies instantaneously. But, still with anonymous objects work can be extracted before it dies like calling a method using the anonymous object:
- Anonymous object instantiation:

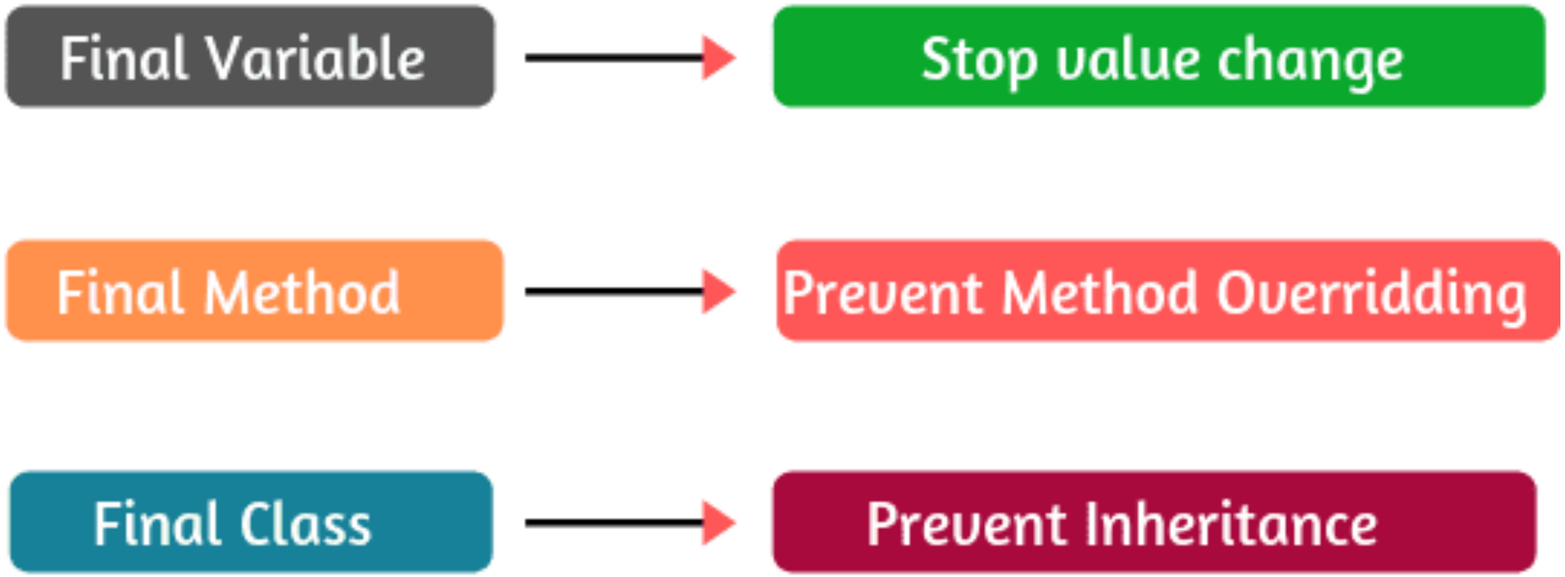
```
new Test()
```

Anonymous Objects

```
class Test{
void fact(int  n){
  int fact=1;
  for(int i=1;i<=n;i++){
   fact=fact*i;
  }
   System.out.println("factorial is "+fact);
}
public static void main(String[] args) {
    //calling method with anonymous object
    new Test().fact(5);
  }
}
```

Final Keyword

Final Keyword

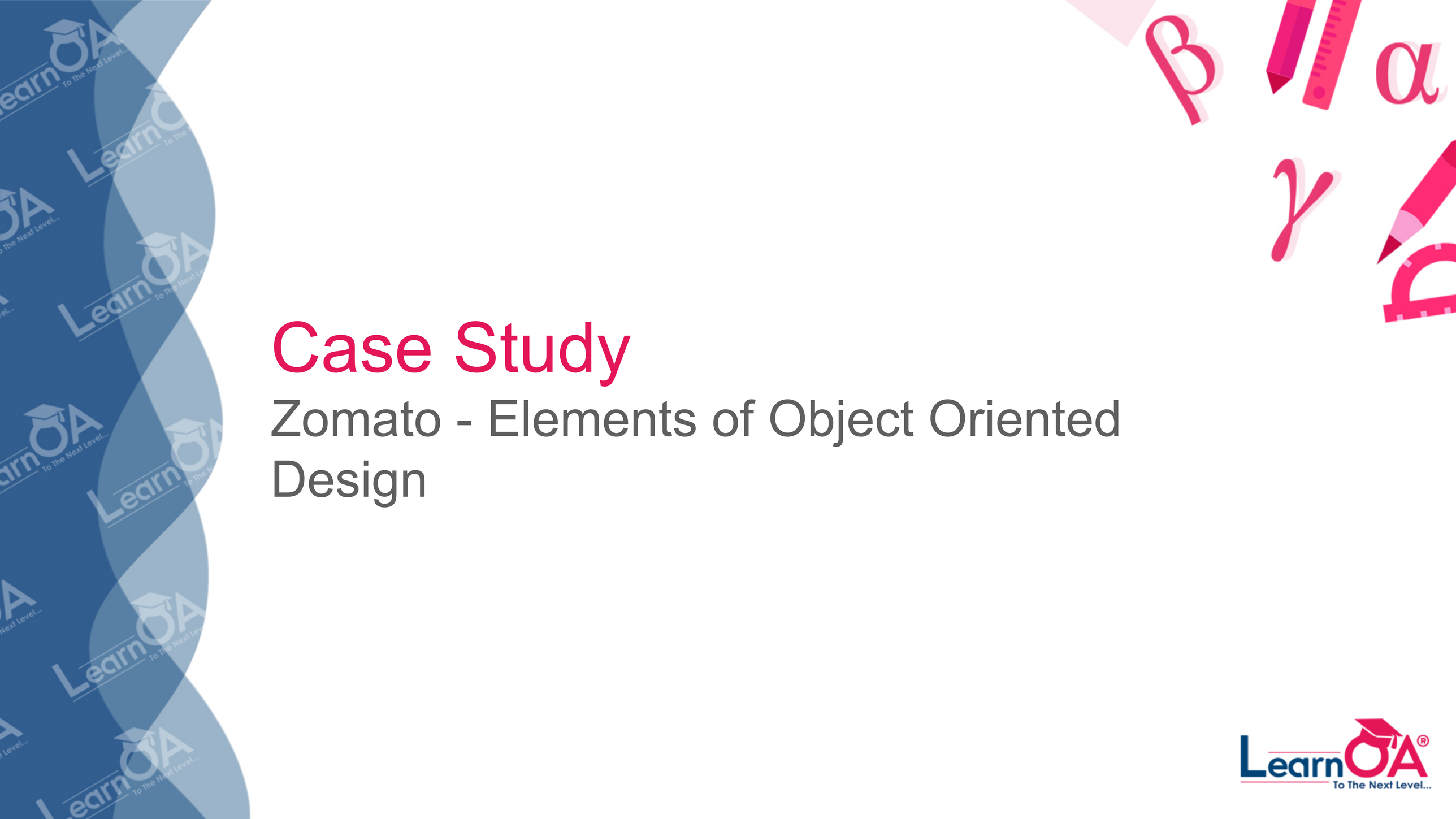


Object Class

Object Class Methods

Root Class in Java. Below are some of important functions to be discussed:

toString()
equals()
hashCode()
getClass()
wait()
notify()
etc..



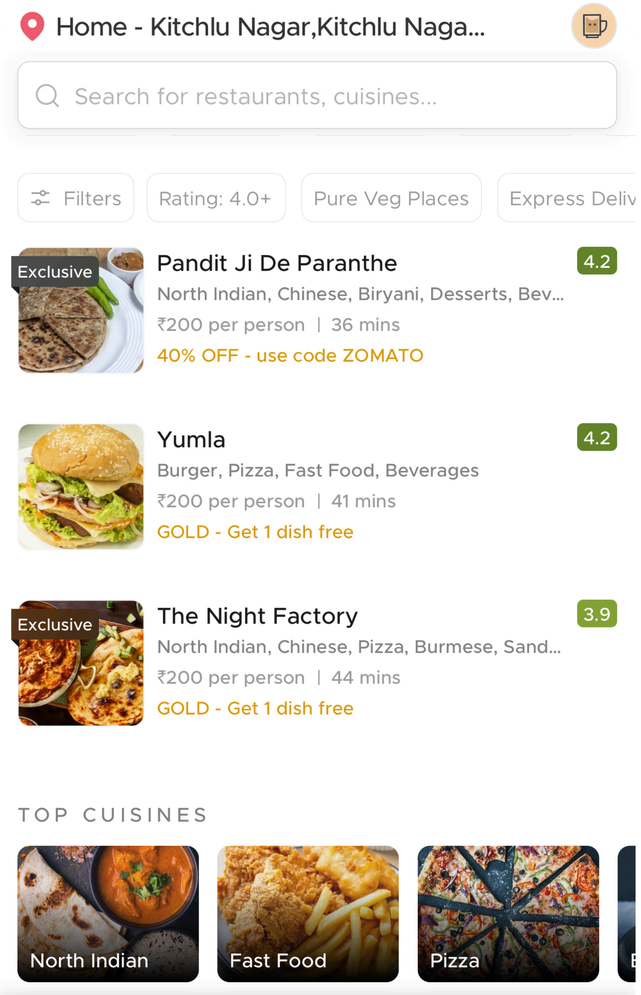
Case Study

Zomato - Elements of Object Oriented Design

Problem

1. Listing Restaurants in Application
2. Identification Of Objects and Relationships between them

Developing Solution



Thank You