About Me:

I am Akash Palghadmal.

Graduated from Fr CRCE, MU in 2019.

Currently working as a Server-side Developer. My Interests lie in Java, Scala and other server side languages.

Co-founder and organizer of UnScript Hackathon.

Grand Finalist of SIH2018.

Lead CodelabsCRCE (technical council) during the year 2017-2019.

If you want to connect to me, the best place would be linkedIn.

POP

- Procedure oriented programing.
- Whole program is treated as a big procedure of instructions.
- Procedures can be divided into smaller chunks which we call methods of functions.
- E.g. C, Pascal, Fortran, etc

OOP

- Object oriented programing.
- Divided into classes/objects
- Program is modularized and integrated.
- E.g. JAVA, C#, etc

Disadvantages of POP:

- Global data access
- Maintenance is pain
- Duplicate method for same functionality and different data types
- One program has to do everything
- Error finding is another level pain

Basics for OOPs:

Data Variable:

- Data variable is used to store some data.
- E.g. int x = 10
  - x <= variable name
  - “Int” is the data type (what it should store. Int means integer)
  - 10 is the value stored in x

Methods/functions:

- Methods are blocks of code that perform certain computation
- public int sum(int x, int y){
  return (x+y)
  }
  This method returns sum of x and y. Let’s break this and understand

Basics for OOPs: (contd)

```
public int sum (int x, int y) {
 // everything inside this is computation of the method
// the return statement returns the result to caller
}
```

public : access modifier (can be applied to variables as well)
int : return type (which kind of data will be returned)
sum : that's the name of method
(int x, int y): parameters
Parameters are defined as type of data and a name for them

Access Modifiers:

1. Public : everywhere
2. Protected: only subclass
3. Private: same class

Basics for OOPs: (contd)

What is class?

- Class is a blueprint.
- E.g. blueprint of a house.
- Class definition will answer the following:
  - What will it have?
  - What will it do?

What is object?

- Object is instance of class.
- E.g. Real house itself.
- Object has following:
  - State
  - Behaviour
  - Identity

Basics for OOPs: (contd)

All objects are created from their corresponding classes.
E.g. of class

```
class A {
        public int x = 10;
        public int square(){
        {
                return (x*x);
        }
}
class Test {
        public static main(String[] args){
                A a = new A()
                print a.square()
        }
}
```

A() <= constructor.
I can write my own or use default one.
If no constructor is mentioned, java will use the default format.
User defined constructor

```
class A{
        A(int x){
                print x;
        }
        ….
}
call<= A a = new A(10);
```

No return types for constructor

Basics for OOPs: Qs

Which of the following statement is true?

A. An Object can exist without a class.
B. A class can exist without an object.
C. Both need to exist mandatorily.

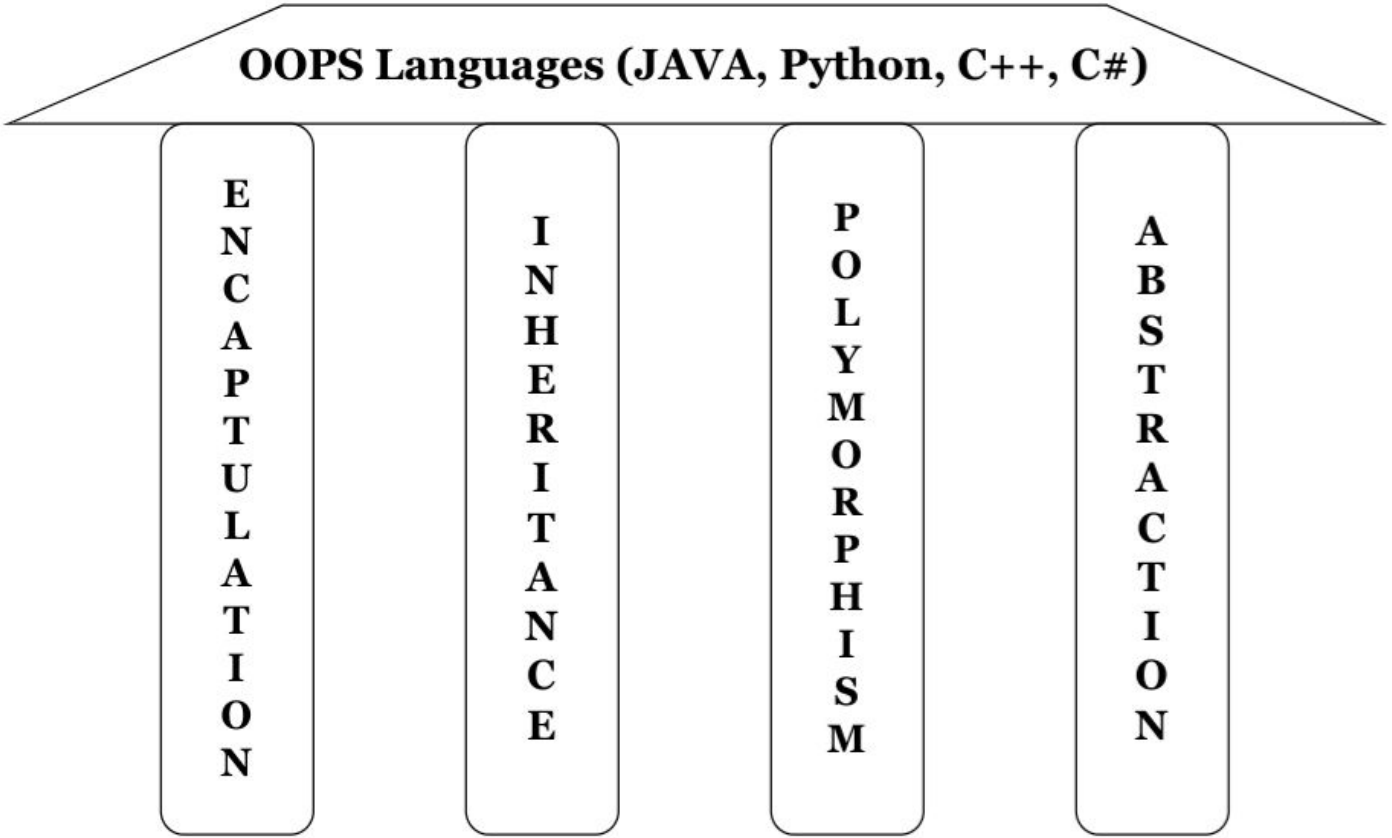
Short Break:

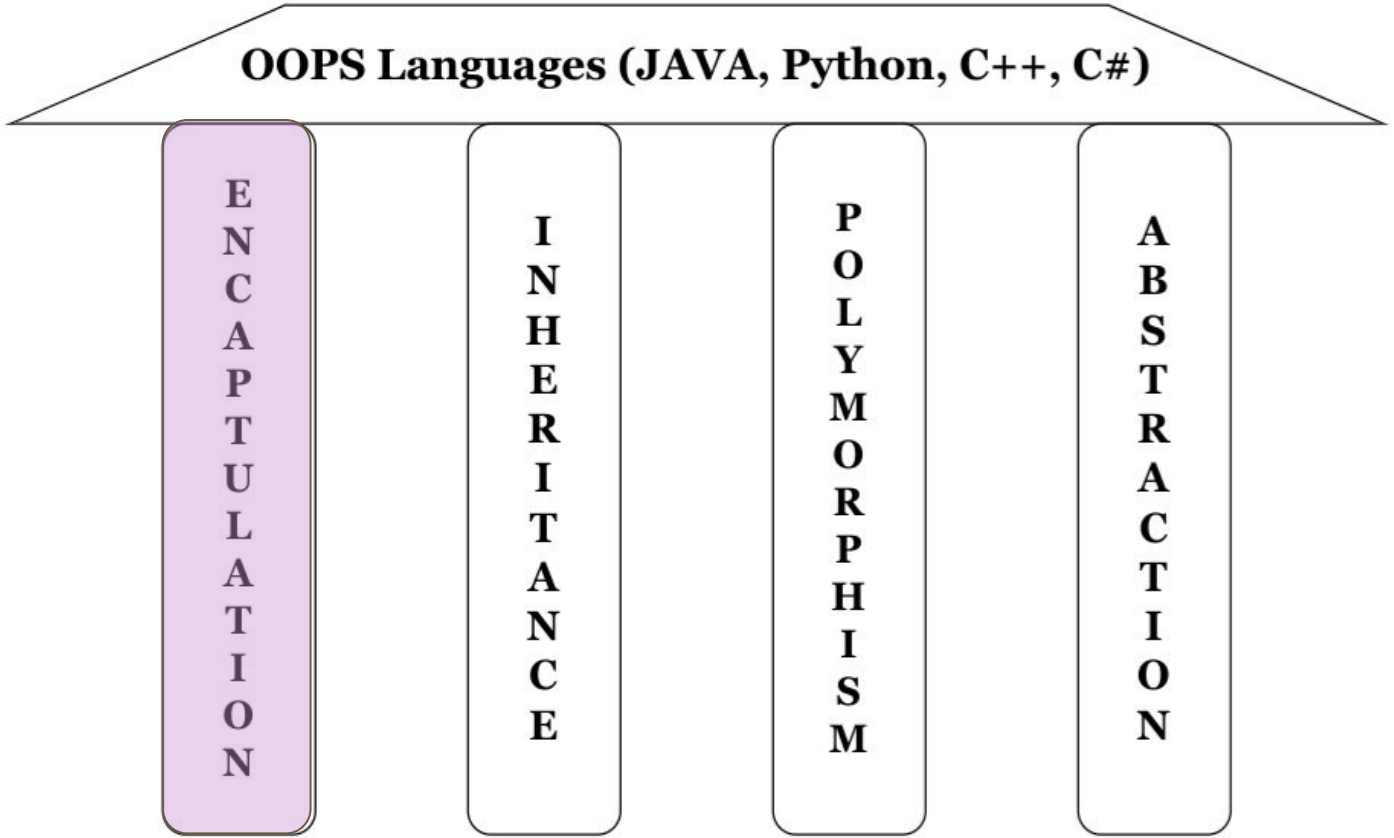
Any Questions till this point?
If not, solve this puzzle.

There are 10 bags of gold with infinite gold coins in them. Every bags has coins of 10 gms each except for one bag. It has all coins of 11 grams. You have a digital weighing Machine with you. Tell me the least number of times You need to use the machine to find of the faulty gold bag?

(this was one of my interview question)

Image source: www.pixabay.com





Encapsulation:

Qs. I have 4 variables of students: Id, name, pointer, year. I want to create Java class/es for them so that I can use this variables. How should I do that?

A. Create 4 classes. One for each variable and store them in their respective classes
B. Create a single class and store all 4 in them

Of course to create a single class with all 4 variables in them

Encapsulation:

This is called Tight Cohesion

Related data and methods that manipulate that data or use that data for computation should be placed together i.e. they should be closely bound.

```
class Student {
        int id;
        String name;
        Double pointer
        String year;
        //whatever methods required
}
```

PEEP:
Place for Everything, Everything in its place

Encapsulation:

Qs. All the students are placed. So FrCRCE decides that each student will receive a reward equivalent to package * pointer* 100, i.e. if package was 10 lacs and pointer was 8.5, then you will get 8,500/- rupees as prize money. You can collect your money from class representative. College has asked you to design a system for CRs to compute prize money for each student. Assume that student ID, name , pointer and package are available in the class Student

A. Create a method in class student that based on Student ID, gets data for package and pointer, computes the result and displays.
B. Create a method in class Student that takes pointer and package and computes and returns the result.

Encapsulation:

Answer: A

Why A?

Data Privacy/ Data Hiding.

As mentioned the class Student had data set as Id, name, pointer and package. So based on their ID i can get data of their pointer and package and then return them the results.

=> Solves the drawback of global data access that we faced in POP

Fun Fact:

WHO SPILLED THE JAVA BEANS?

Image Source: www.pixabay.com

JAVA Beans:

- JAVA Beans are also known as POJO
- POJO: Plain Old Java Object
- Which basically means to create an object that stores data of an entity and reads and writes on that data.

```
class Student {
        private int id;
        private String name;
        private double pointer;
        private String year;

        public int getId(){
                return this.id;
        }
        public void setId(int id){
                this.id = id;
        }
        //similar get and set methods for rest of the variables
        //this methods are called getter and setter methods
}

//Main class
Student s = new Student();
s.setId(7657)
print s.id    //will get me compile time error as private variables are not accessible outside of class they are declared
print s.getId()  //this will work
```

Encapsulation: Summary

Encapsulation revolves around 2 things:

1. Data hiding/ Data privacy.
2. Tight Cohesion

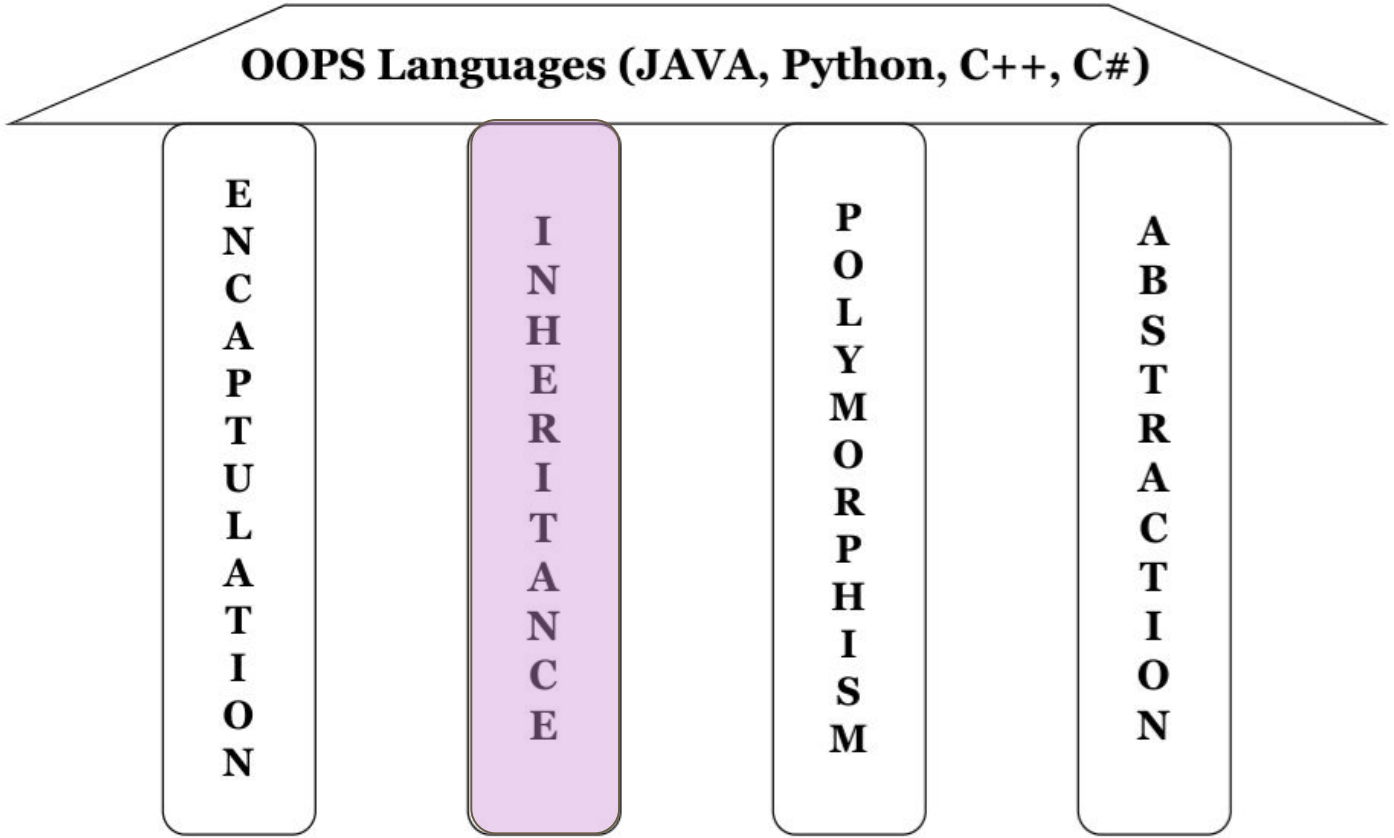
Short Break:

Any Questions till this point?
If not, solve this puzzle.

On day 1 a man starts climbing up a hill at 8 am and reaches the top at 6 pm. Next day he start descending at 8 am and reaches the foot at 6pm. His speed is inconsistent and he might be taking breaks at random interval.
On these two day will there be a moment when this Man will be at same location at same time
i.e say at time X on day 1 he is at point Y and time x of day 2 he is again at point Y?

(this was one of my interview question)

Image source: www.pixabay.com



Inheritance: Introduction and Simple Inheritance

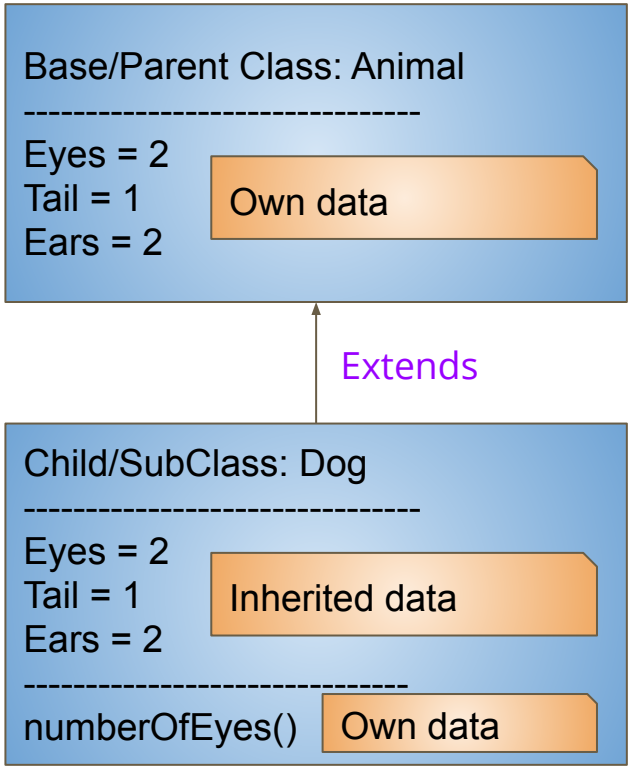
- Inherit : Take characteristics from someone/something
- In OOP we use the word “extends” to do this
- E.g :

```
Class A {
        int x = 10
        int square() {
                return x*x;
        }
}
Class B extends A {
        void printsquare() {
                println( square() );
        }
}
```

Simple Inheritance

- Extends A basically tells that B inherits all the properties that A has.
- Which basically means that all the variables in and methods in A are accessible from B.
- In our example B can access variable x and method square() without creating an object of A.

Inheritance: Simple Inheritance



Lets understand with an example:

```
class Animal{
        int eyes = 2
        int tail = 1
        int ears = 2
}
Class Dog extends Animal {
        void numberOfEyes(){
                Print eyes;
        }
}
```

Inheritance: Multilevel Inheritance

```
Class A {
        …
}
```

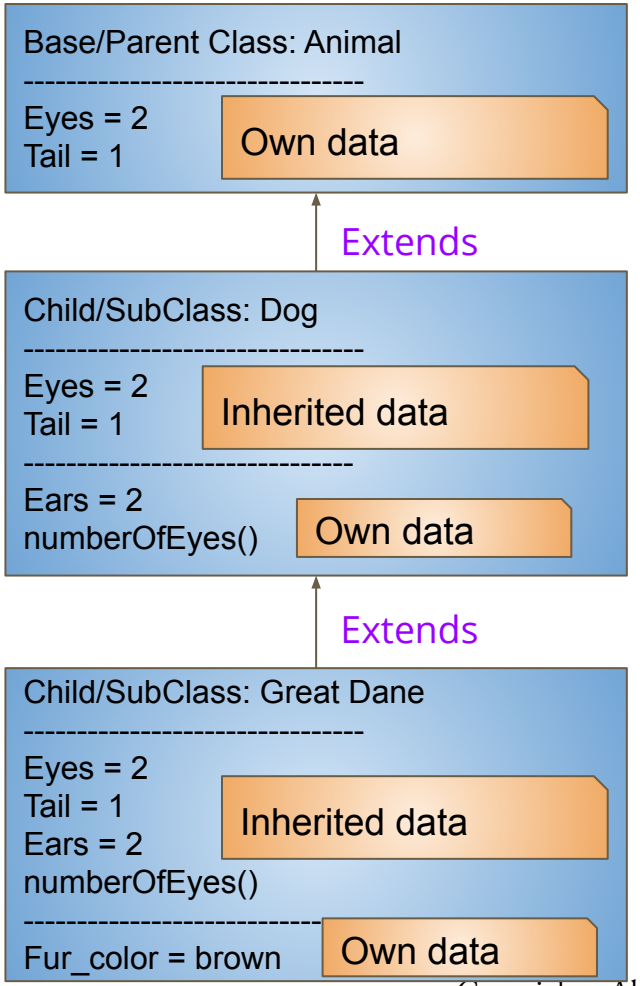
Can use all variables and methods of A

```
Class B extends A{
        …
}
```

Can use all variables and methods of B as well as A

```
Class C extends B {
        …
}
```

Can use all variables and methods of C and B and A



Lets understand with an example:

```
class Animal{
        int eyes = 2
        int tail = 1
}
Class Dog extends Animal {
        int ears = 2
        void numberOfEyes(){
                Print eyes;
        }
}
Class Great_Dane extends Dog {
        int fur_color = brown;
}
```

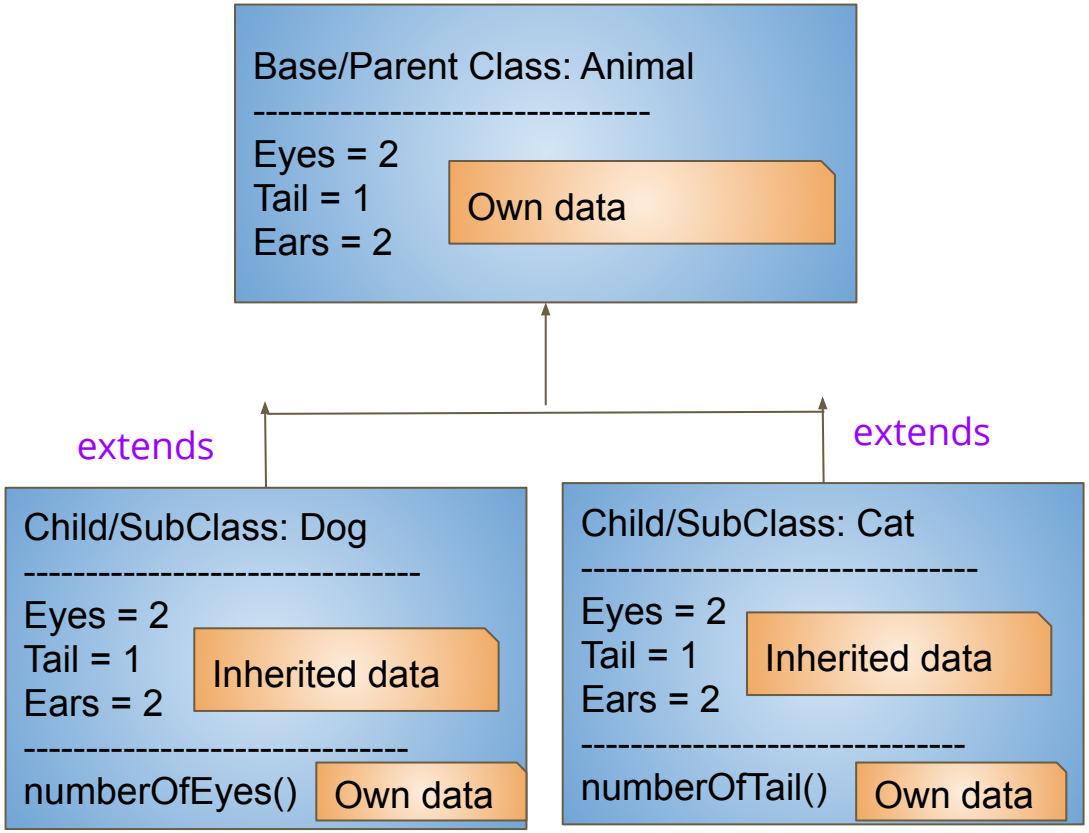
Inheritance: Hierarchical Inheritance

```
class A {
        …
}
class B extends A {
        …
}
class C extends A {
        …
}
```

Can use all variables and methods of A

Can use all variables and methods of B as well as A

Can use all variables and methods of C and A but not B



Lets understand with an example:

```
class Animal{
        int eyes = 2
        int tail = 1
        int ears = 2
}
Class Dog extends Animal {
        void numberOfEyes(){
                Print eyes;
        }
}
Class cat extends Animal {
        void numberOfTail(){
                Print tail
        }
}
```

Inheritance: Multiple Inheritance

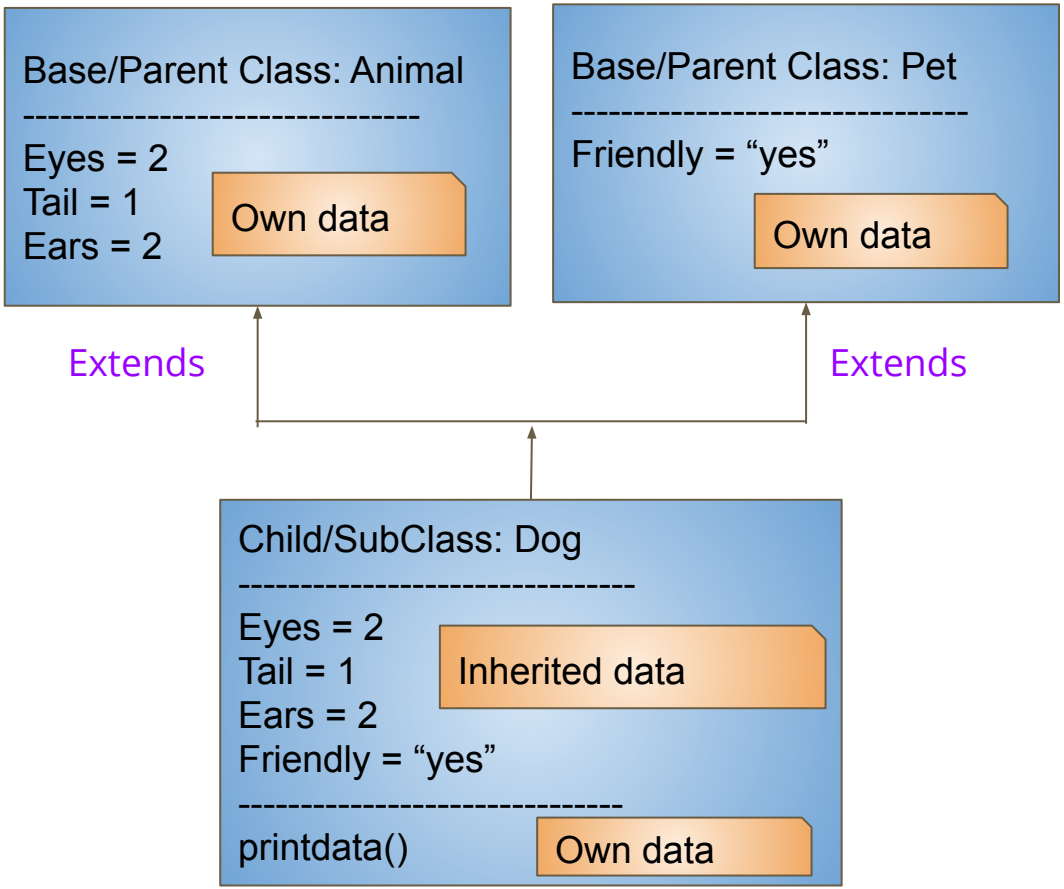
```
class A {
        …
}
class B {
        …
}
class C extends A,B {
        …
}
```

Can use all variables and methods of A

Can use all variables and methods of B

Can use all variables and methods of C as well as A and B

Only Supported in C++.
JAVA, C#, etc do not support this.



Lets understand with an example:

```
class Animal{
        int eyes = 2
        int tail = 1
        int ears = 2
}
class Pet{
        String friendly = “yes”
}
Class Dog extends Animal, Pet {
        void printdata(){
                Print eyes;
                Print friendly;
        }
}
```

Inheritance: Scope of variables and methods

- Public: Accessible from subclass
- Protected: Accessible from subclass
- Private: Not accessible from subclass
- Default (no modifier): Not accessible from subclass

Inheritance: Scope of variables and methods

```
Class A {
        int x =10
        private int y =10
        public int z =10
        protected int w =10
}
Class B extends A {
        … // only z and w will be accessible in B, while x and y if used will throw compile time error
}
```

Inheritance: static and dynamic binding

class Vehicle {...}
class Car extends Vehicle{...}

Is_a relationship is nothing but inheritance.

How to create an object of class B?

1. Car car = new Car();
   This is called static binding. At compile time machine will know that type of object is Car and the class called is also Car
2. Vehicle car = new Car();
   This is called dynamic binding. At compile time machine will be confused whether the instance will be Vehicle or Car. but since Car extends Vehicle this is allowed as Car is_a Vehicle

Inheritance: Question

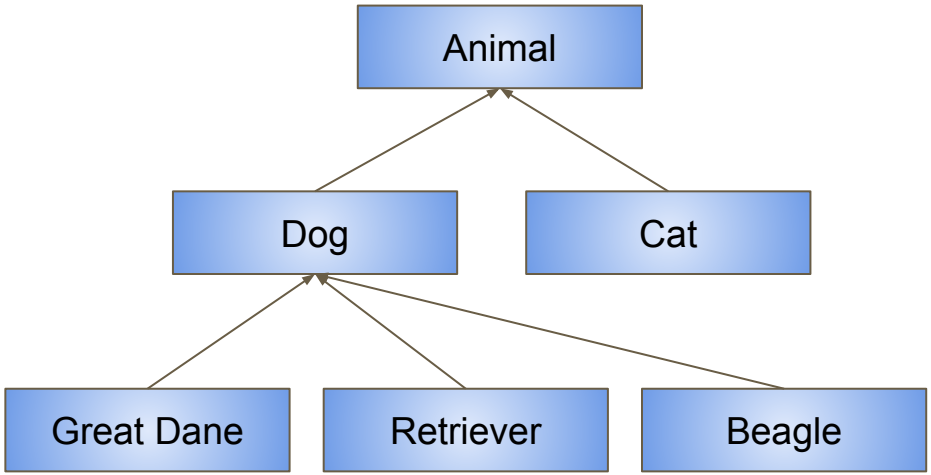
class Vehicle {...}
class Car extends Vehicle{...}

Car car = new Vehicle();
If I write this statement in the code what will happen?

1. It will run successfully.
2. It will throw error at compile time.
3. It will throw error at run time.

Vehicle is_a car?
No
This compiler also knows.
So error will be thrown at compile time saying
“They are incompatible types”

Inheritance in nutshell



Dog is a Animal

Cat is a Animal

Cat is a Dog

Beagle is a Dog

Beagle is a Animal

Beagle is a Cat

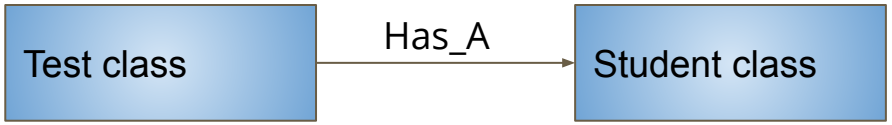
Do cat and great dane have anything in common?

Yes.
The properties of animal are common in both of them

Difference between composition and Inheritance

```
class Student {
        public int x = 10;
        public int square(){
        {
                return (x*x);
        }
}
class Test {
        public static main(String[] args){
                A a = new A()
                print a.square()
        }
}
```

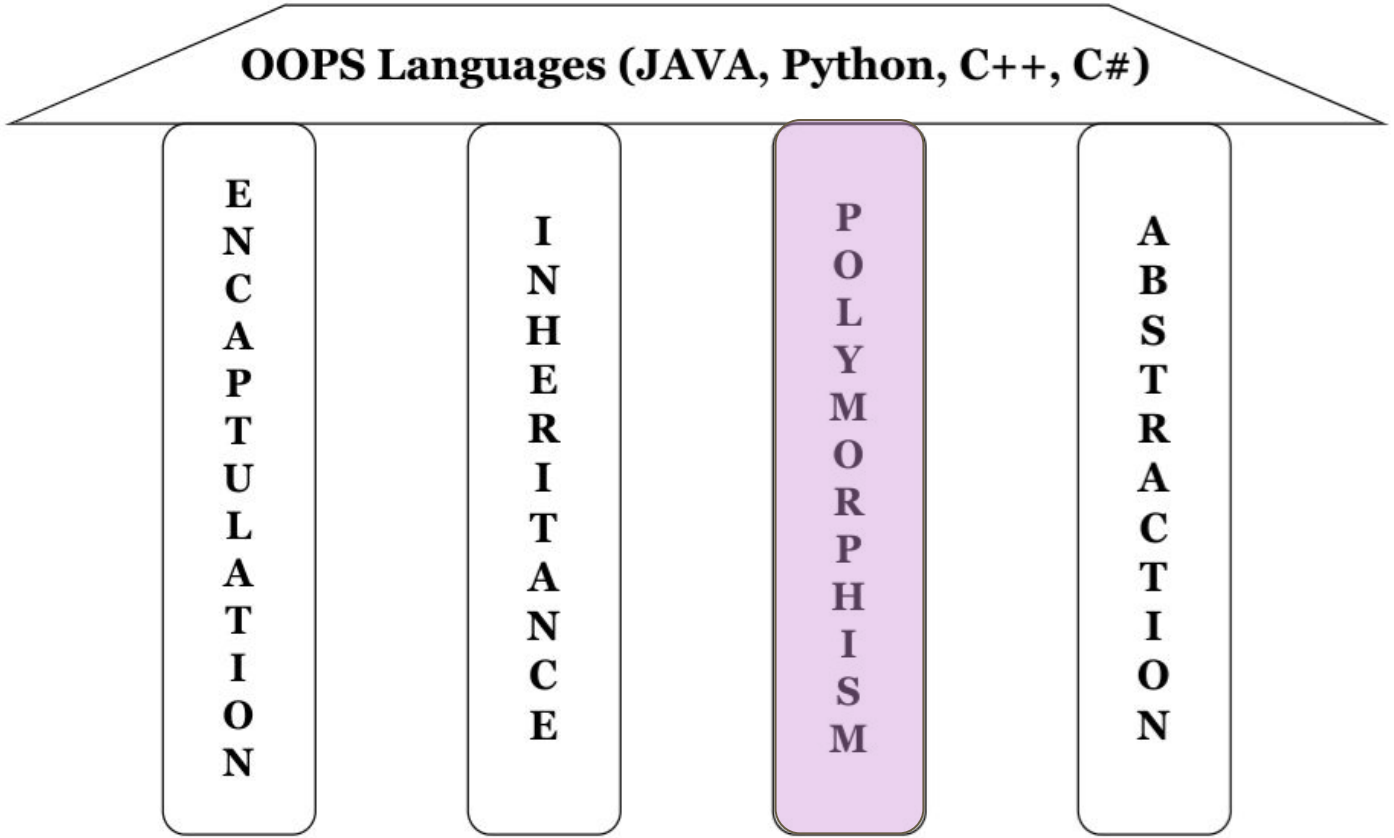
Composition: creates an instance
Inheritance: extends parent



Composition is has_a relationship.
Test has_a Student

Inheritance is is_a relationship.
Dog is_a Animal

Composition means object is used inside another class.
Inheritance means child-parent relationship, where child inherits all data/methods of parents.



Polymorphism: Introduction

Duplicate method names for same functionality.

Poly = many; morph = form

=> many forms

Same method name to carry out different thing.

There are 2 ways to do it:

1. Overloading
2. Overriding

This solves the duplicate method names for same functionalities and different data types issue

Polymorphism: Overloading

Rules for Overloading:

- Same class
- Same method name
- Different arguments/parameters.
- Return type does not matter

Polymorphism: Overloading

Eg 1:

```
int sum (int x, int y)
{
        return x + y
}
double sum (double x, double y)
{
        return x + y
}
```

======

sum(1,2); //this will go to first one

sum(1.0,2.0); //this will go to second one

Polymorphism: Overloading

```
void sum (int x, int y)
{
        Print x + y
}
Void sum (double x, double y)
{
        print x + y
}
```

Same class

Same method name

Different arguments/parameters

Return type doesn't matter

Polymorphism: Overloading

```
Void sum (int x, int y)
{
        Print x + y
}
Void sum (int x, int y, int z)
{
        Print x + y + z
}
```

Same class

Same method name

Different arguments/parameters

Return type doesn't matter

Polymorphism: Overloading

```
Int sum (int x, int y)
{
        Print x + y
}
Double sum (int x, int y)
{
        Print x + y
}
```

Same class

Same method name

Different arguments/parameters

Return type doesn't matter

Polymorphism: Overriding

Rules for Overriding:

- Inherited Classes
- Same method name
- Same arguments/parameters.
- Same return type

Polymorphism: Overriding

```
Class A {
        int sum(int x, int y){
                return (x+y)
        }
}
Class B extends A {
        int sum(int x, int y){
                return (x*x+y*y)
        }
}
```

Inherited class

Same method name

Same arguments/parameters

Same return type

Polymorphism: Overriding


```
Class A {
        int sum(int x, int y){
                return (x+y)
        }
}
Class B extends A {
        int sum(int x, int y, int z){
                return (x+y+ z)
        }
}
```

Inherited class

Same method name

Same arguments/parameters

Same return type

Polymorphism: Overriding

```
Class A {
        int sum(int x, int y){
                return (x+y)
        }
}
Class B extends A {
        void sum(int x, int y){
                print (x+y)
        }
}
```

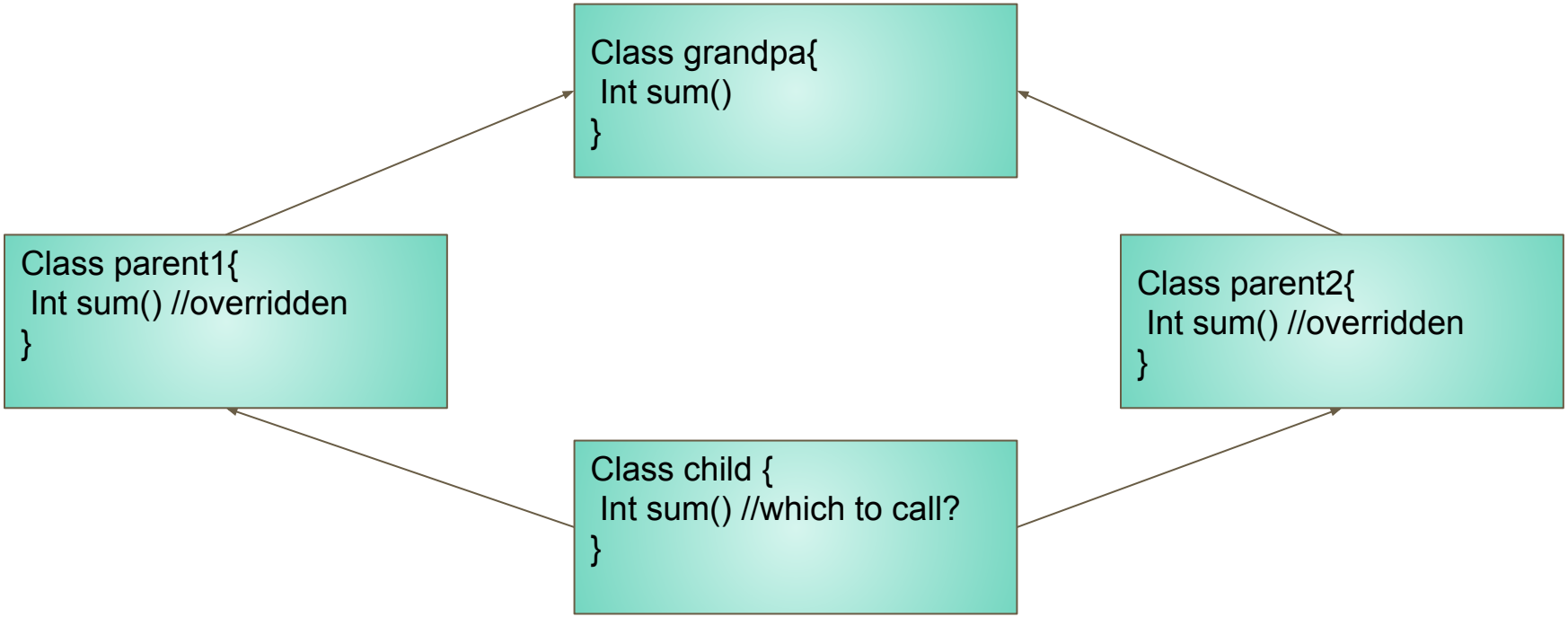
Inherited class

Same method name

Same arguments/parameters

Same return type

Multiple Inheritance: Diamond Problem



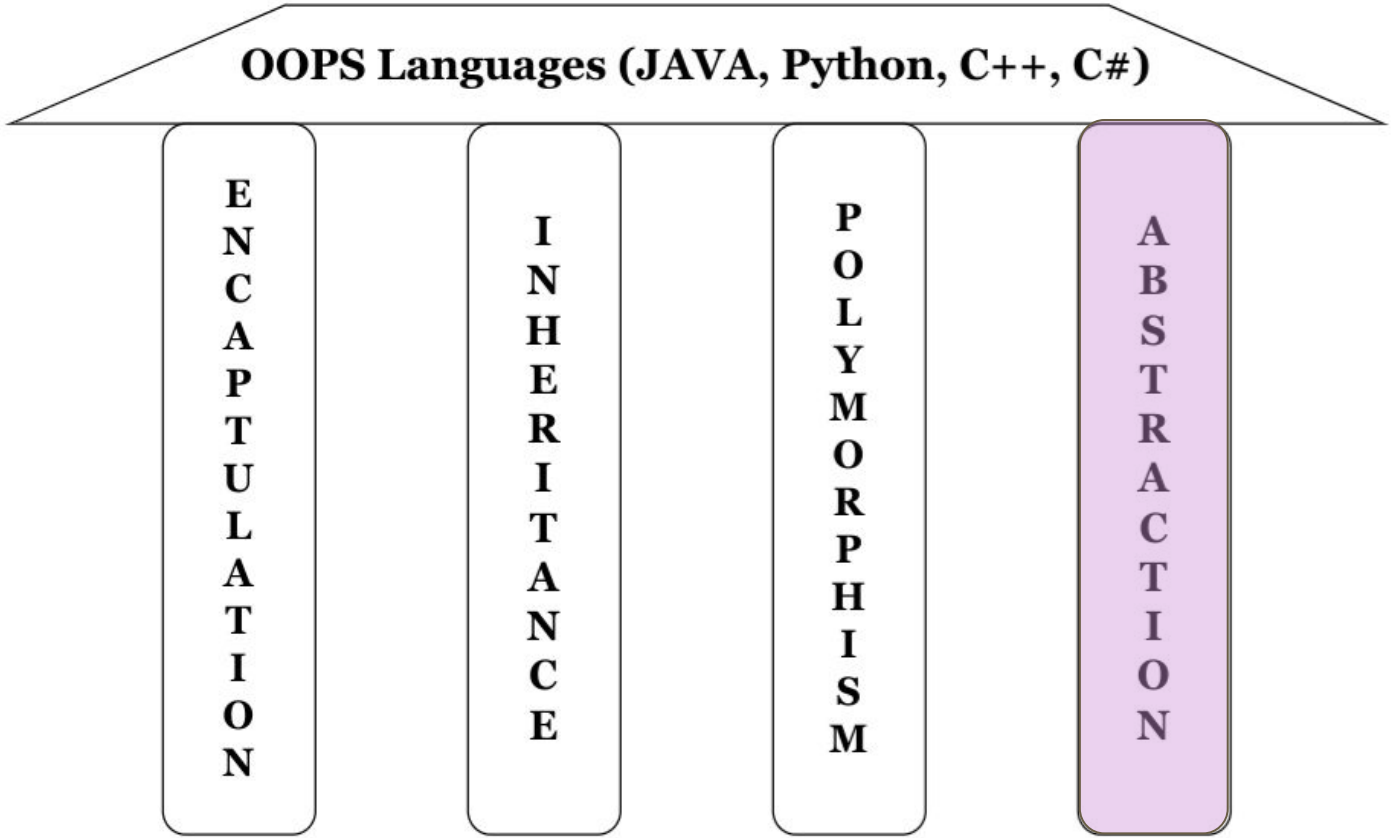
Short Break:

Any Questions till this point?
If not, solve this puzzle.

You have 8 balls, in which one of them is either heavy or light weight and rest all are same weight. You have a weight balance with you. What is the minimum number of attempts it will take you to find the faulty ball.

(this was one of my interview question)

Image source: www.pixabay.com



Abstraction: Introduction

- Abstraction means data hiding.
- Only that data should be visible to users which is required by them.
- This addresses data privacy issue.
- Sections:
  - Abstract class and abstract method.
  - Interfaces
  - Solution to diamond problem

Abstraction: Abstract class and abstract methods

How to create an abstract class?

1. abstract class A {
   ....
   }
2. There should be at least 1 abstract method inside abstract class.
   abstract int sum(int x, int y);
3. You cannot create object of abstract class. You have to extend it via non abstract class to use its properties.
4. The non abstract class should implement all abstract methods of A. Sum in our example

Abstraction: Abstract class and abstract methods

```
abstract class A {
        int sum(int x, int y);
}

class B extends A {
        int sum(int x, int y){
                return (x+y);
        }
}
```

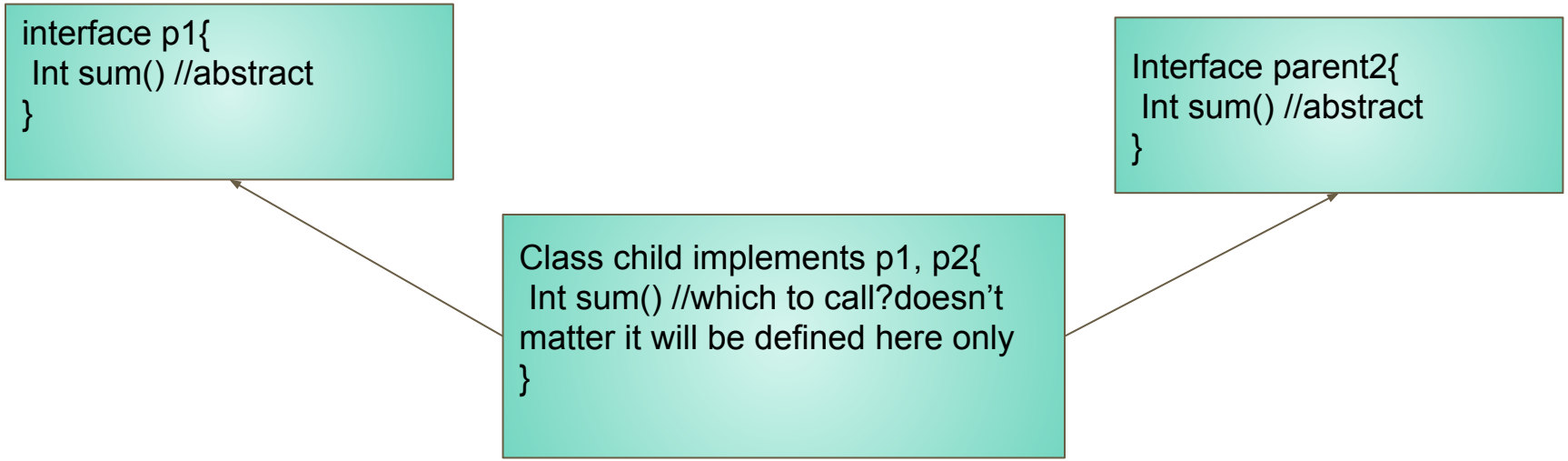
I can create an object of B, but not of A

Abstraction: Interface

- Idea of Interface came with abstract class
- In interface all the methods are abstract.
- All the variables are “public static final” type. Which basically means they are constant.
- E.g.

```
interface MyInterface{
  public void method1();
  public void method2();
}
class Demo implements MyInterface{
  public void method1(){
        System.out.println("implementation of method1");
  }
  public void method2(){
        System.out.println("implementation of method2");
  }
}
```

Multiple Inheritance: Diamond Problem Solved



Abstraction: Question

I have two things Animal and Walk.For say class Dog which should be abstract class and which should be Interface.

A. Both Interface
B. Both Abstract class
C. Animal abstract class Walk Interface
D. Walk Abstract class Animal Interface

This question is from coding standards

C.
Dog is_a Animal (Inheritance -> extends -> abstract class)
Dog can Walk (ability or behavioural quality must be interface)

Multi-threading: Overview

- Multi-threading means dividing a large computational task into smaller versions of n tasks to complete it faster.
- E.g. calculating sum of 1- 1000 numbers can be sped up if
  We create 10 threads that compute 1-100,101-200,.. Simultaneously.
- In Java, multithreading can be achieved using Thread class or Runnable Interface.
- Threads can be created or killed inside the application based on use cases.

This solves the one program has to do everything issue of POP

Multi-threading: Interesting fact

There is class which has to call an external source of third party library. It is found that over the time this library starts leaking memory (internal issues which we cannot control). How to take care of this?
Ans: We create a thread and call this source from there, when it starts leaking we just kill the thread and start a new one.

This question was asked to me and Edwin for one of our interviews.

Q and A section:

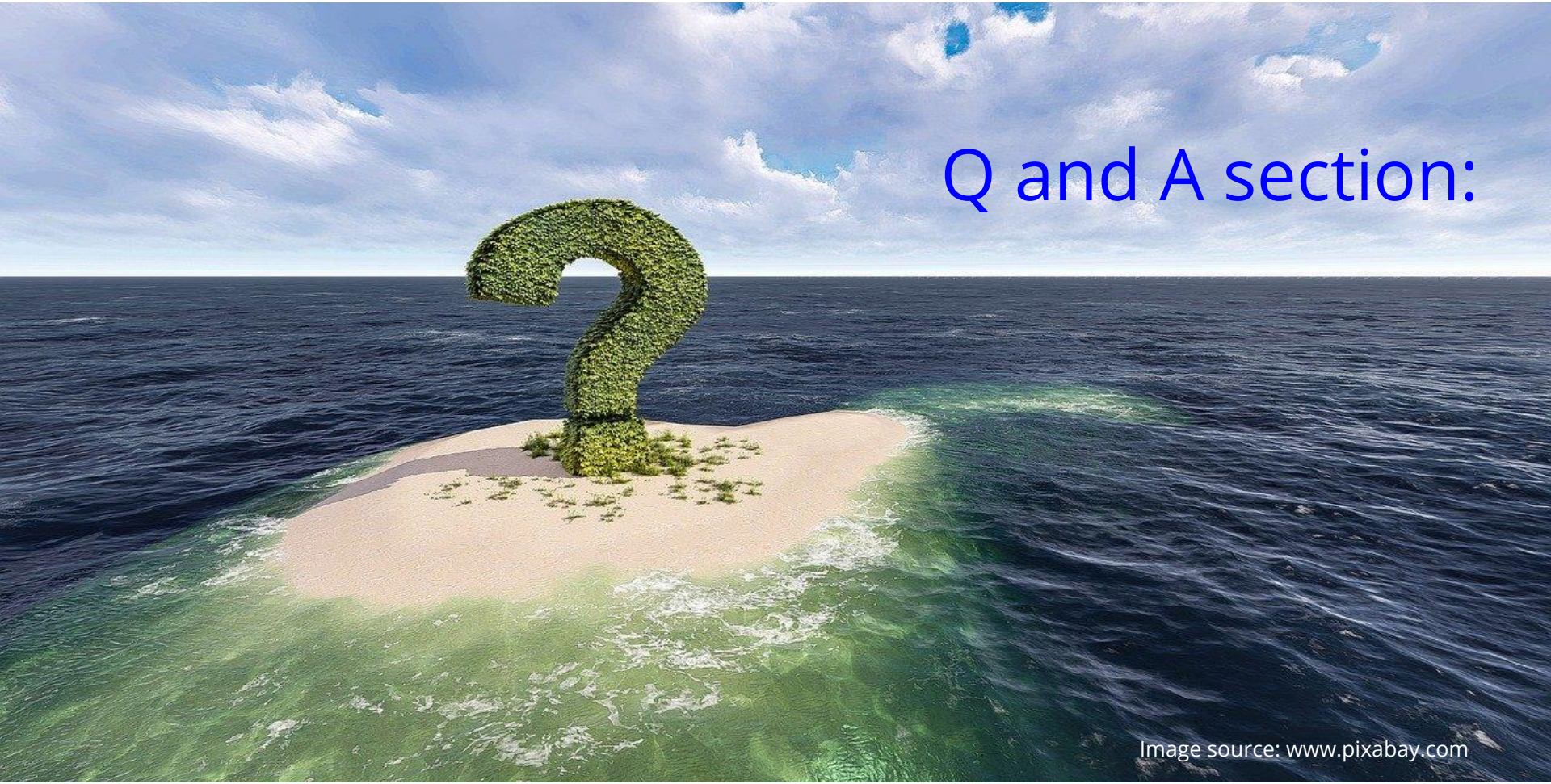


Image source: www.pixabay.com

References:

- Geeks for Geeks : https://www.geeksforgeeks.org/object-oriented-programming-oops-concept-in-java/
- Journal Dev: https://www.journaldev.com/12496/oops-concepts-java-example
- JavaTPoint: https://www.javatpoint.com/java-oops-concepts

Note: This is not an exhaustive list, but these are the sites i use to refresh my OOPS concepts knowledge. A better way to make your own notes while studying and then to refer them before interviews.

Image source: www.pixabay.com