Interface

— Continuation

Difference b/w Abstract class & Interface

Abstract class

→ In case of abstract class we can have concrete methods

→ we can have final methods

→ It is compulsory to declare abstract keyword for abstract methods

→ we can have instance variables

→ we can have constructor

→ we can have instance & static block

→ we cannot achieve multiple inheritance

Interface

→ Cannot have concrete methods

→ Cannot have final methods

→ by default it is abstract no need to declare explicitly

→ we cannot have instance variables [static only]

→ By default they are public static final

→ we cannot have constructor

→ we cannot have instance & static block

→ we can achieve multiple inheritances

① Need of Constructor in abstract class ?

Ⓐ

```
abstract class Parent {
    int a;
    int b;
    Parent(int a, int b) {
        this.a = a
        this.b = b
    }
    void play();
}
```

```
class Child extends Parent{
    int c;
    int d;
    Child(int a, int b, int c, int d) {
        super(a,b);
        this.c = c;
        this.d = d;
    }
```

→ to initialize the variables which are inherited from parent class

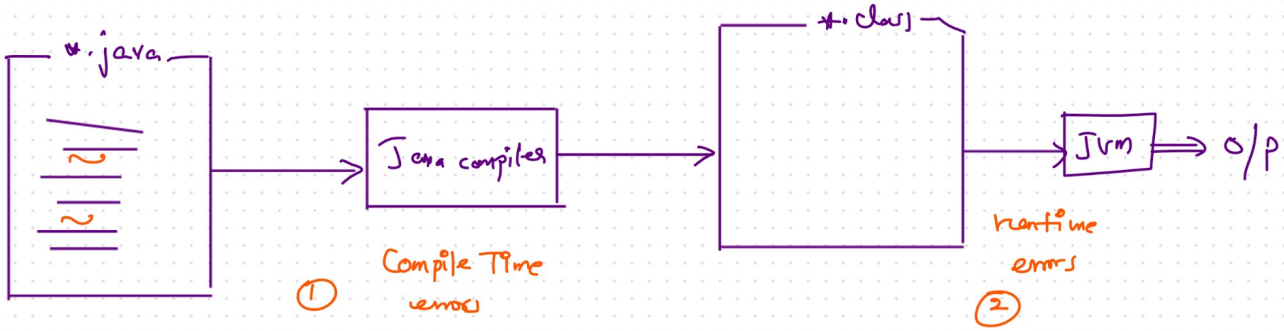
→ We can achieve Constructor chaining

4 pillars of OOP

1) Encapsulation
2) Inheritance
3) polymorphism
4) Abstraction

Exception Handling

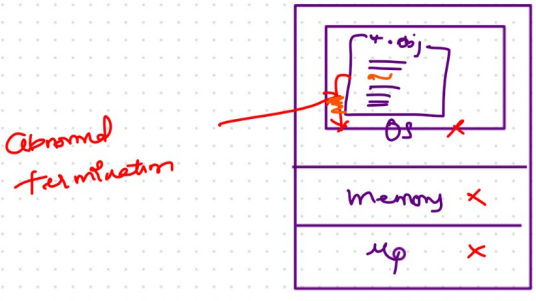
Is it Necessary to rectify the errors ?



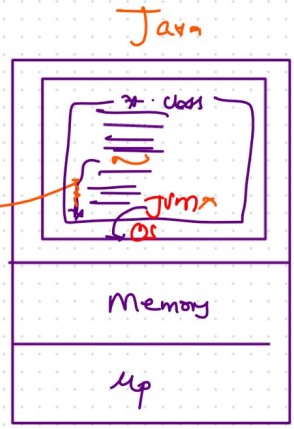
∴ It is compulsory to handle the errors

Case 1:

C language.



Case 2:



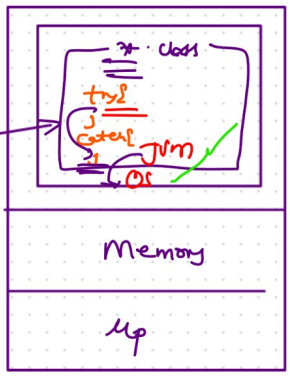
Abnormal termination

JVM handles it & gives the control normal back to OS

but the execution is not as expected

Abnormal Termination

Case 1:



developed user the execution handling

→ So therefore errors are handled in runtime & given control back to JVM normally & JVM gives back control to OS also normally

Normal Termination

Exception Handling

Def: Exception is a unwanted or unexpected event which occurs during the program execution, which disturbs the normal flow of the execution

→ the main objective of exception handling is for the graceful termination of the program

→ Exception handling is a way of writing the alternative code to continue the normal flow of program

try {

Risky code

}

catch (*** e) {

alternate code to handle exception

}

```
public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter a, b values:");
        int a = sc.nextInt();
        int b = sc.nextInt();
        try {
            System.out.println(a/b);
        }catch (ArithmeticException e) {
            System.out.println(e);
            System.out.println("B value cannot be zero");
        }
        System.out.println("Program completed...");
}
```

Note:

→ In the above program there is a every possibility, the exception may occur in the line S.O.P(a/b), when ever user enters the value of b = 0

→ If the exception occurs, the program would teminat abnormally

→ To overcome this problem we need to handle the exception

→ the exception can be handled with the help of

try, catch block

→ where ever if there is an exception [thrown] put that code / line like [S.O.P(a/b)] in the try block

→ Write the alternative solution in the catch block to handle the exception

→ the catch block always follows the try block

→ If any exception occurs inside the try block, the control would directly go to the catch block

→ after the execution of the catch block, the program flows normally