* Polymorphism *

(many forms)

Poly → many

morphism → forms

defn: An entity which behaves differently in different cases is called as polymorphism.

It enables us to do one action in many different ways.

* Types of Polymorphism
  1. Compiletime (static polymorphism) : Ex: method overloading, method hiding.
  2. Runtime (dynamic polymorphism) : Ex: method overriding, dynamic dispatch.

- Polymorphism is an ability of an object to behave differently at different situations.

Ex: . Power Button has Polymorphic behaviour.
. + operator in java

- Polymorphism provides Flexibility,
- Inheritance " Reusability of code.
- Encapsulation " Security of data.

* Static Polymorphism

- Static polymorphism can be achieved using method overloading and Hiding.
- Instance method overloading and for static it's method Hiding.
- Java compiler is responsible to bind the method calls with actual method at compile time.

method overloading

You can write multiple methods inside the class with same name by changing parameters, this process is called method overloading.

Rules for method overloading

1. methodname must be same
2. method parameters must be changed in terms of
   - a. Number of parameters
   - b. Type of parameters
   - c. order of parameters
3. method return type can be of any type.

. Method overloading exists in the same class

. method overloading is not for changing any business logic but just for increasing the maintainability and readability of the code.

```
class Arithmatic {

    void sum (int a){
        sop (a+a);
    }
    void sum (double a) {
        sop (a+a);
    }
    void sum (double a , double b) {
        sop (a+b);
    }
    int sum (int a , int b) {
        return a+b;
    }
    void sum (int a , double b) {
        sop (a+b);
    }
```

```
void sum (double a, int b) {
    sop (a+b);
  }

}

public class mainclass {

  psvm (-) {

  Arithmetic al = new Arithmatic ();
    al.sum (10);
    al.sum (10.5);
    al.sum (10.5, 20.5);
    al.sum (10, 20.9);
    al.sum (10.5, 20);
    al.sum (10, 10);

  }

}
```

Runtime polymorphism :

Method overridding

- According to object oriented best practices "classes are closed for modification". i.e If you want to modify the functionality of existing class then you should not distrub existing class.

- It is always better to write a subclass and provide the required implementations in sub class.

- Subclass can be written for following purposes.
  - To add new functionality
  - To modify existing functionality.
  - To inherit existing functionality.

Rules for overridding

(1) Subclass method name should be same as super class method name.

(2) subclass and superclass method parameters (type, order, number) must be same.

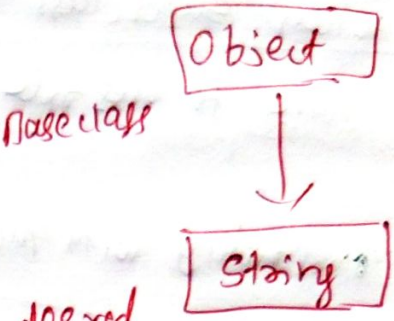
(3) Subclass method return type must be same as super class method return type.

[[return type of super class method is of type class then in subclass you can use same or it's subclass as return type]]

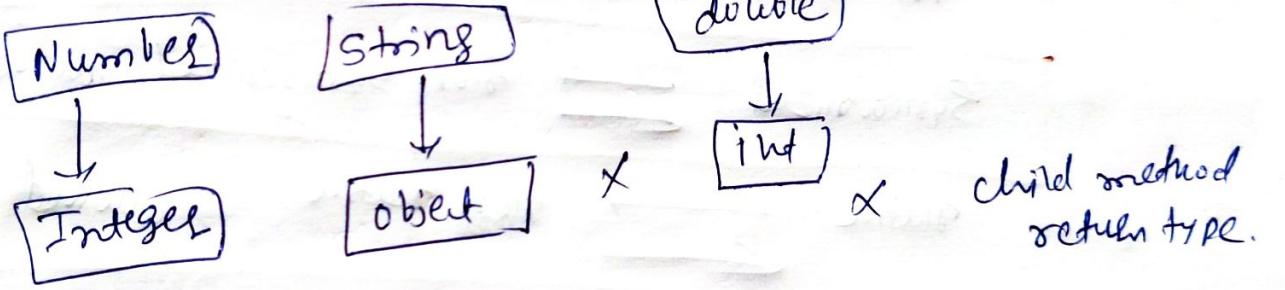
i.e co-variant return type.

Ex:

```
class P
{
  public object m1()
  {
    return null;
  }
}
class C extends P
{
  public String m1()
  {
    return null;
  }
}
```



Ex:



(4) Subclass method access modifier must be same or higher then super class method access modifier

Super class method	Subclass method
Public	Public
Protected	Protected, Public
default	default, Protected, Public
Private	Private, default, Protected, Public

(5) When Super class method is instance method then you have to override in subclass as instance only.

(6) When superclass method is static method then you have to override in subclass as static only.

(7) overriding with modifiers

Parent class method	child class method	
final	→ non final / final	Not allowed
abstract	⇄ non abstract	allowed
non-final	→ final	allowed
Synchronized	⇄ non Synchronized	allowed
native	⇄ non native	allowed
StrictFP	⇄ non-StrictFP	allowed
static	→ static	allowed
non static	→ non static	allowed

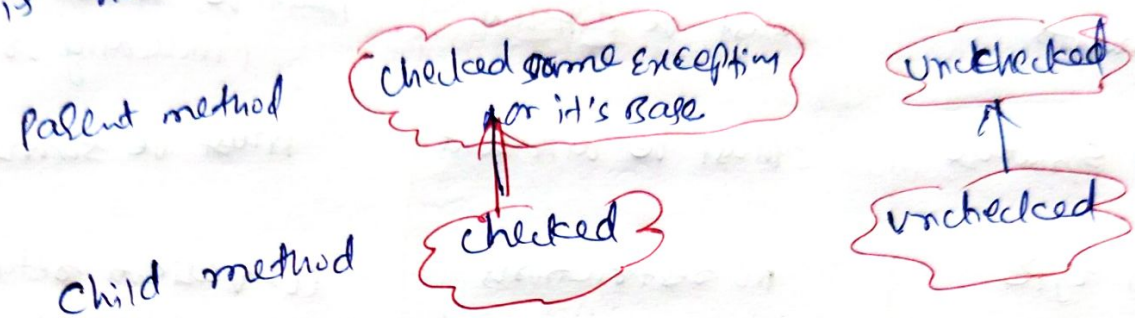
(8) We can't override final method, But can be used as it is.

(9) Private methods are not getting inherited in subclass So, overridding concept not applicable.

(10) This rule is applicable while implimenting Interface methods.

(11) If child class method throws some checked exception then compulsory parent class method should throw the same checked exception or it's parent class exception, otherwise CTE

There is no rule for unchecked Exception



(12) overriding concept is not applicable for variable

Example program

```
class A {
 void show () {
sop("I am from A class show");
   }
}
class B extends A {
 void show() {
  sop ("I am from B class show");
    }
}
```

```
class Runclass {
 psvm(-)
  {
B b = new B();
  b.show();
   }
}
```

Difference Between overloading and overriding

Property	overloading	overriding
1. method name	Same	Same
2. arguments	Must be different	must be same (including order)
3. method signature	Must be different	must be same
4. return type	No restrictions	Co-variant return type are allowed.
5. Private, static & final methods	overloaded	can't overridden
6. Access modifiers	No restrictions	can't decrease the scope.
7. Exception	No restrictions	No restriction for unchecked exception. But checked exception if child throws then parent should throw same or it's parent Exception
8. method resolution	Based on reference type	Based on runtime object
9. occurance	within same class	Between superclass and subclass.
10. Also known as	Compile time Polymorphism Static Polymorphism early binding	Runtime polymorphism dynamic polymorphism Late binding

Method Hiding

The Rules of method hiding all are same as of overriding But Except following.

1. Both derived class method and Base class method should be static
2. method resolution Based on Reference type
3. it is considered as compile time Polymorphism, or early Binding.

Example :→

```
class Base {
    Public static void m1()
    {
        sop("parent");
    }
}
class child extends Base {
    public static void m1()
    {
        sop("child");
    }
}
class Test {
    psvm(-)
    {
        Base b = new Base();
        b.m1();   ← Parent called
        child c = new child();
        c.m1();   ← child called
```

Method Hiding — Must be static

```
Base bc = new
    Base();
bc.m1();
```
↑ parent called

Dynamic Dispatch:

Dynamic dispatch is the process of assigning subclass object to superclass reference variable.

```
Base b = new Derived();
```

↑ Base class reference

↑ Derived class object