Q) What do you mean by POLYMORPHISM in Java? What are the different types of Polymorphism? What are the advantages & disadvantages of Polymorphism?

Polymorphism :- Performing single action (behavior → method) in many different ways.

Polymorphism = many + form

Two types:

- Compile time or Static Polymorphism : method overloading
- Runtime or Dynamic Polymorphism : method over-riding : Dynamic method Dispatch

Advantages

- Code reusability is the main advantage of polymorphism; once a class is defined, it can be used multiple times to create an object.
- In compile-time polymorphism, the readability of code increases, as nearly similar functions can have the same name, so it becomes easy to understand the functions.
- The same method can be created in the child class as in the parent class in runtime polymorphism.
- Easy to debug the code. You might have intermediate results stored in arbitrary memory locations while executing code, which might get misused by other parts of the program. Polymorphism adds necessary structure and regularity to computation, so it is easier to debug.

Disadvantages

- Implementing code is complex because understanding the hierarchy of classes and its overridden method is quite difficult.
- Problems during downcasting because implicitly downcasting is not possible. Casting to a child type or casting a common type to an individual type is known as downcasting.
- Sometimes, when the parent class design is not built correctly, subclasses of a superclass use superclass in unexpected ways. This leads to broken code.
- Runtime polymorphism can lead to the real-time performance issue (during the process), it basically degrades the performances as decisions are taken at run time because, machine needs to decide which method or variable to invoke.

Q) What do you mean by Compile-time / static polymorphism or method overloading in Java. Give some real world examples.

↳ two or more methods in the same class with same function-name, and different argument list

Rules for Overloading

1. Number of arguments should be different
   or
2. Order of arguments should be different
   or
3. Type of argument should be different

Change in return type does not matter!

⇒ determined by compiler : which function call to be binded with which method definition.


class Sum {
public void sum(int a, int b) {
System.out.println(a + b);
}
// Number of Arguments
public void sum(int a, int b, int c) {
System.out.println(a + b + c);
}
// Datatypes of Arguments
public void sum(String a, String b) {
System.out.println(a + b);
}
public void sum(String a, int b) {
System.out.println(a + b);
}
// Order of Arguments
public void sum(int a, String b) {
System.out.println(a + b);
}
}
def^n body
class Driver {
Run | Debug
public static void main(String[] args) {
Sum obj = new Sum();
obj.sum(a: 10, b: 15);
obj.sum(a: 5, b: 10, c: 15);
obj.sum(a: "Archit", b: "Aggarwal");
obj.sum(a: "Archit's score: ", b: 100);
obj.sum(a: 100, b: " is Archit's score");
}
}
call (invokation)
25
30
ArchitAggarwal
Archit's score: 100
100 is Archit's score
// Compilation Error: Variable names
// does not matter
// public void sum(int c, int d){
// }
// Compilation Error: Return type does
// not matter
// public int sum(int a, int b){
// return a + b;
// }
Re-declaration Error (Compilation)
overloading

```java
class User {
    String name;

    public void setName(String firstName, String
    middleName, String lastName) {
        name = firstName + " " + middleName + " " +
        lastName;
    }

    public void setName(String firstName, String
    lastName) {
        name = firstName + " " + lastName;
    }

    public void setName(String firstName) {
        name = firstName;
    }

    public void setName(char firstLetter, char
    secondLetter, String lastName) {
        name = firstLetter + ". " + secondLetter +
        ". " + lastName;
    }
}
```

```java
User u1 = new User();
u1.setName(firstName: "Sachin",
middleName: "Ramesh", lastName: "Tendulkar");
System.out.println(u1.name);

User u2 = new User();
u2.setName(firstName: "Virat",
lastName: "Kohli");
System.out.println(u2.name);

User u3 = new User();
u3.setName(firstName: "Tejas");
System.out.println(u3.name);

User u4 = new User();
u4.setName(firstLetter: 'K',
secondLetter: 'L', lastName: "Rahul");
System.out.println(u4.name);
```

Real World Example

```
Sachin Ramesh Tendulkar
Virat Kohli
Tejas
K. L. Rahul
```

8) What do you mean by run-time/dynamic polymorphism or method overriding in Java. Give some real world examples.

↳ One method in parent class, another method in child class, with same function prototype

- same function name
- same return type *(TRC conditions)
- same argument list.
  - same no of arguments
  - same order of arguments
  - datatype of arguments same

In runtime, JVM binds the function call to the corresponding function definition

There must be inheritance for over-riding, ie. two methods in different classes related as parent-child.

```java
class Parent {
    private int a, b;

    public int getA() {
        return a;
    }

    public void setA(int a) {
        this.a = a;
    }

    public int getB() {
        return b;
    }

    public void setB(int b) {
        this.b = b;
    }

    // overriden method
    public void printObject(int a, int b) {
        System.out.println(x: "Parent's Object : ");
        this.setA(a);
        this.setB(b);
        System.out.println(this.getA() + " " + this.getB());
    }
}
```

```java
class Child extends Parent {
    private int p, q;

    public int getP() {
        return p;
    }

    public void setP(int p) {
        this.p = p;
    }

    public int getQ() {
        return q;
    }

    public void setQ(int q) {
        this.q = q;
    }

    // overrided method
    public void printObject(int p, int q) {
        System.out.println(x: "Child Object : ");
        this.setP(p);
        this.setQ(q);
        System.out.println(super.getA() + " " + super.getB());
        System.out.println(this.getP() + " " + this.getQ());
    }
}
```

```java
public static void main(String[] args) {
    Parent obj1 = new Parent();
    obj1.printObject(a: 10, b: 20);
    // overrident method -> Parent

    Child obj2 = new Child();
    obj2.printObject(p: 40, q: 50);
    // overrided method -> Child

}
```

```
Parent's Object :
10 20
Child Object :
0 0
40 50
```

```java
class User {
    String name;
    String location;

    public User(String name, String location) {
        this.name = name;
        this.location = location;
    }

    // overriden method
    public void bookShow() {
        System.out.println(this.name + " " + this.location);
        System.out.println(x: "Error: You cannot book the show");
        System.out.println(x: "Please, first login or sign up");
    }
}
```

```java
class RegisteredUser extends User {
    String emailId;
    long phoneNo;

    public RegisteredUser(String name, String location, String emailId,
    long phoneNo) {
        super(name, location);
        this.emailId = emailId;
        this.phoneNo = phoneNo;
    }

    // overrided method
    public void bookShow() {
        System.out.println(super.name + " " + super.location + " " +
        this.emailId + " " + this.phoneNo);
        System.out.println(x: "Please select the number of seats");
        System.out.println(x: " And proceed to payment gateway");
    }
}
```

```java
User u1 = new User(name: "Archit", location: "Delhi");
u1.bookShow();

RegisteredUser u2 = new RegisteredUser(name: "Archit",
location: "Delhi", emailId: "archit@gmail.com",
phoneNo: 9319117889l);
u2.bookShow();
```

```
Archit Delhi
Error: You cannot book the show
Please, first login or sign up
Archit Delhi archit@gmail.com 9319117889
Please select the number of seats
 And proceed to payment gateway
```

```
class Child extends Parent {
    private int p, q;

    public int getP() {
        return p;
    }

    public void setP(int p) {
        this.p = p;
    }

    public int getQ() {
        return q;
    }

    public void setQ(int q) {
        this.q = q;
    }
```

```
Parent's Object :
10 20
Parent's Object :
40 50
```

} If child does not have overriding method

Q) What are the differences between method overloading and method overriding?

Method Overloading

(1) implements compile time polymorphism → by javac compiler binding

(2) methods are statically binded (method call determined at compile time)

(3) methods must be in same class

(4) methods must have different argument list.

(5) Return type may or may not be same

Method Over-riding

(1) implements runtime polymorphism → by JVM binding

(2) methods are dynamically binded (method call determined at run-time)

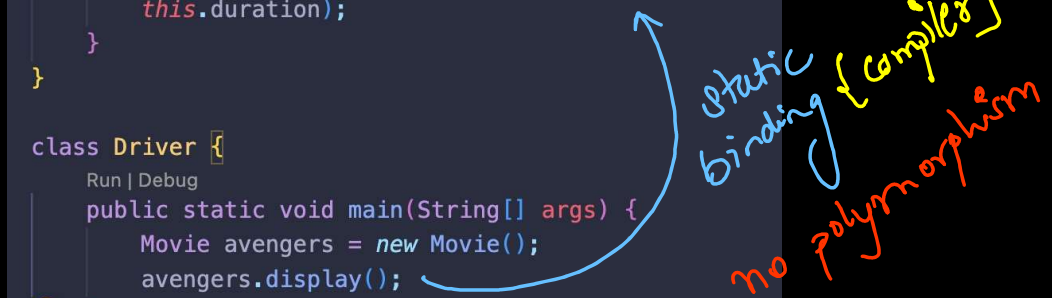
(3) methods must be in super & sub class

(4) methods must have same signature (same return type & argument-list)

(5) Return type (*) must be same → co-variant type

Q) What do you mean by static binding. What do you understand by dynamic binding. What are the differences?
What do you mean by dynamic method dispatch?

```java
class Movie {
    int duration = 180;
    String name = "Avengers Endgame";

    public void display() {
        System.out.println(this.name + " runs for " +
        this.duration);
    }
}

class Driver {
    Run | Debug
    public static void main(String[] args) {
        Movie avengers = new Movie();
        avengers.display();
    }
}
```



Static Binding

→ binding of function call to the corresponding function definition at compile-time by compiler is known as static binding.

→ It is done for methods which are not over-rided and may/may be overloaded.

→ Also known as early binding

→ Static, final, private methods are always statically binded.

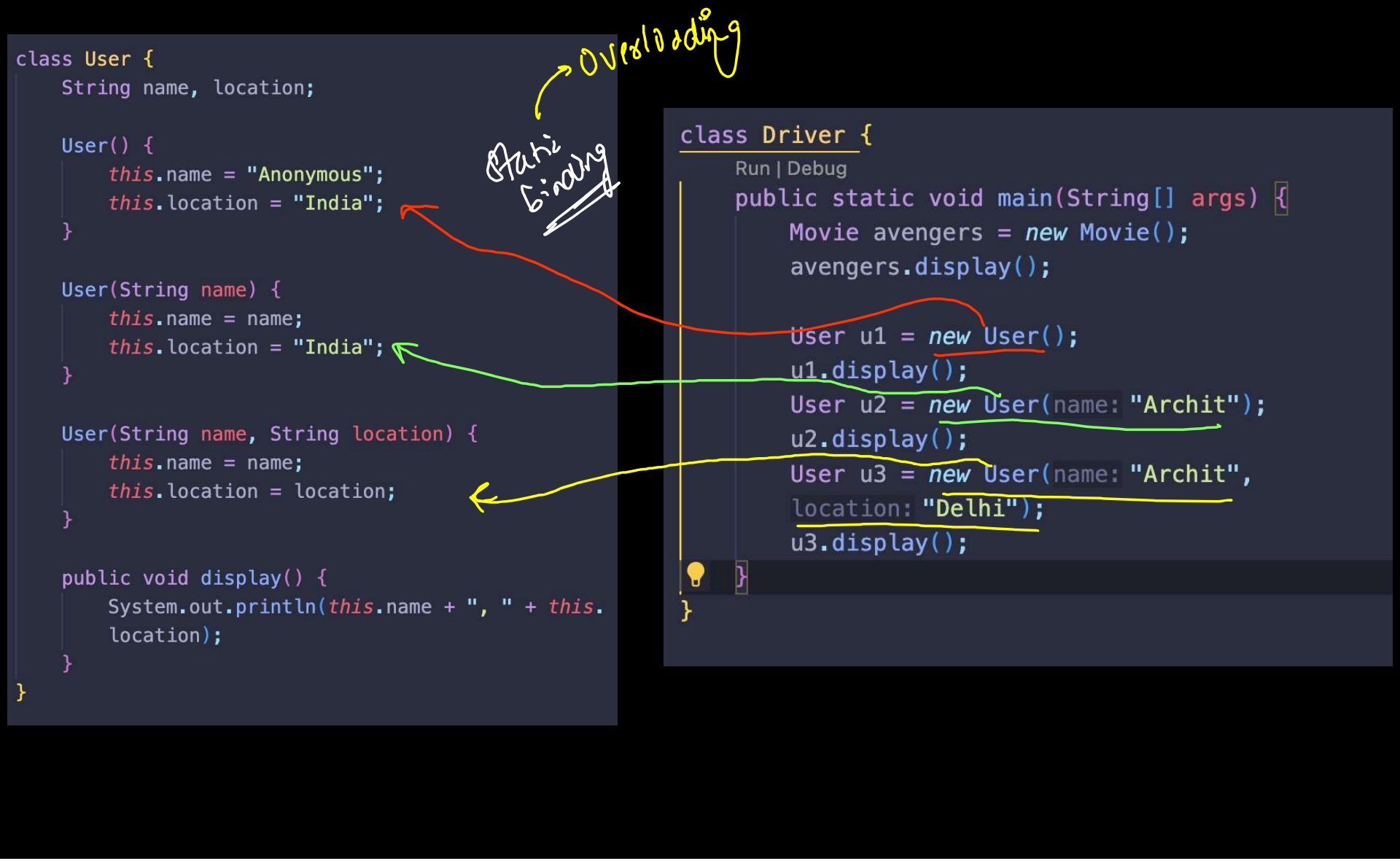
Dynamic Binding

→ binding of function call to the corresponding function definition at runtime by JVM is known as dynamic binding.

→ It is done for methods which are over-rided in the child class → dynamic method dispatch!

→ Also known as late binding

→ Abstract methods are always late binded.


Overloading
Static binding
class User {
String name, location;
User() {
this.name = "Anonymous";
this.location = "India";
}
User(String name) {
this.name = name;
this.location = "India";
}
User(String name, String location) {
this.name = name;
this.location = location;
}
public void display() {
System.out.println(this.name + ", " + this.
location);
}
}
class Driver {
Run | Debug
public static void main(String[] args) {
Movie avengers = new Movie();
avengers.display();
User u1 = new User();
u1.display();
User u2 = new User(name: "Archit");
u2.display();
User u3 = new User(name: "Archit",
location: "Delhi");
u3.display();
}
}

```java
class RegisteredUser extends User {
    String phone = "9319117889";

    @Override
    public void display() {
        System.out.println(this.name + ", " + this.
        location + ", " + this.phone);
    }
}
```

dynamic binding of runtime polymorphism
& overriding

```java
RegisteredUser u4 = new RegisteredUser();
u4.display();
```

```java
class A{
    public void earlyBind(){
        System.out.println("Early Bind");
    }

    public void lateBind(){
        System.out.println("Late Bind in Parent Class");
    }
}

class B extends A{
    @Override
    public void lateBind(){
        System.out.println("Late Bind in Child Class");
    }
}

public class Main{
    public static void main(String[] args){
        A obj = new B();
        obj.earlyBind();→ Early
        obj.lateBind();→ Child
    }
}
```

Q) What are other forms of polymorphism in Java?

- Implicit operator overloading
- Coersion (implicit or explicit type conversion)

We cannot explicitly achieve operator overloading in Java

- complex number addition

c1 = 5+3i
c2 = 7+9i
} ⊕ ✗
add() ✓

```java
public static void main(String[] args) {
    int var1 = 100;
    int var2 = 200;

    System.out.println(var1 + var2);
    // + => Addition of two integers

    String str1 = "Archit";
    String str2 = "Aggarwal";
    System.out.println(str1 + str2);
    // + => Concatenation of String

    System.out.println(var1 + var2 + str1);
    // Addition then Concatenation

    System.out.println(str1 + var1 + var2);
    // Concatenations only
    System.out.println(var1 + str1 + var2);

    System.out.println(x: "\t Archit \n Aggarwal");
}
```

+ operator overloaded

```
300
ArchitAggarwal
300Archit
Archit100200
100Archit200
        Archit
 Aggarwal
```

\ escape sequences → overloading

```java
public static void fun(long var) {
    System.out.println(var);
}
```

Coercion

```java
fun(var: 999999999999l); // long -> long
fun(var: 200); // int -> long
fun(var: 'A'); // char -> long
```

} implicit type conversion
↳ upcasting
smaller data → big data

```java
public static void fun2(char var) {
    System.out.println(var);
}
```

```java
fun2((char) 100l); // long -> char
fun2((char) 35); // int -> char
fun(var: 'A'); // char -> char
```

} explicit type conversion
↳ downcasting
big data → smaller data

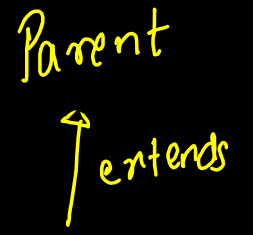
Q) Which of the following are correct declarations. Why or why not?

(a) `Parent obj1 = new Parent();` valid

(b) `Child obj2 = new child();` valid

* (c) `Parent obj3 = new child();` valid

(d) `Child obj4 = new Parent();` not valid

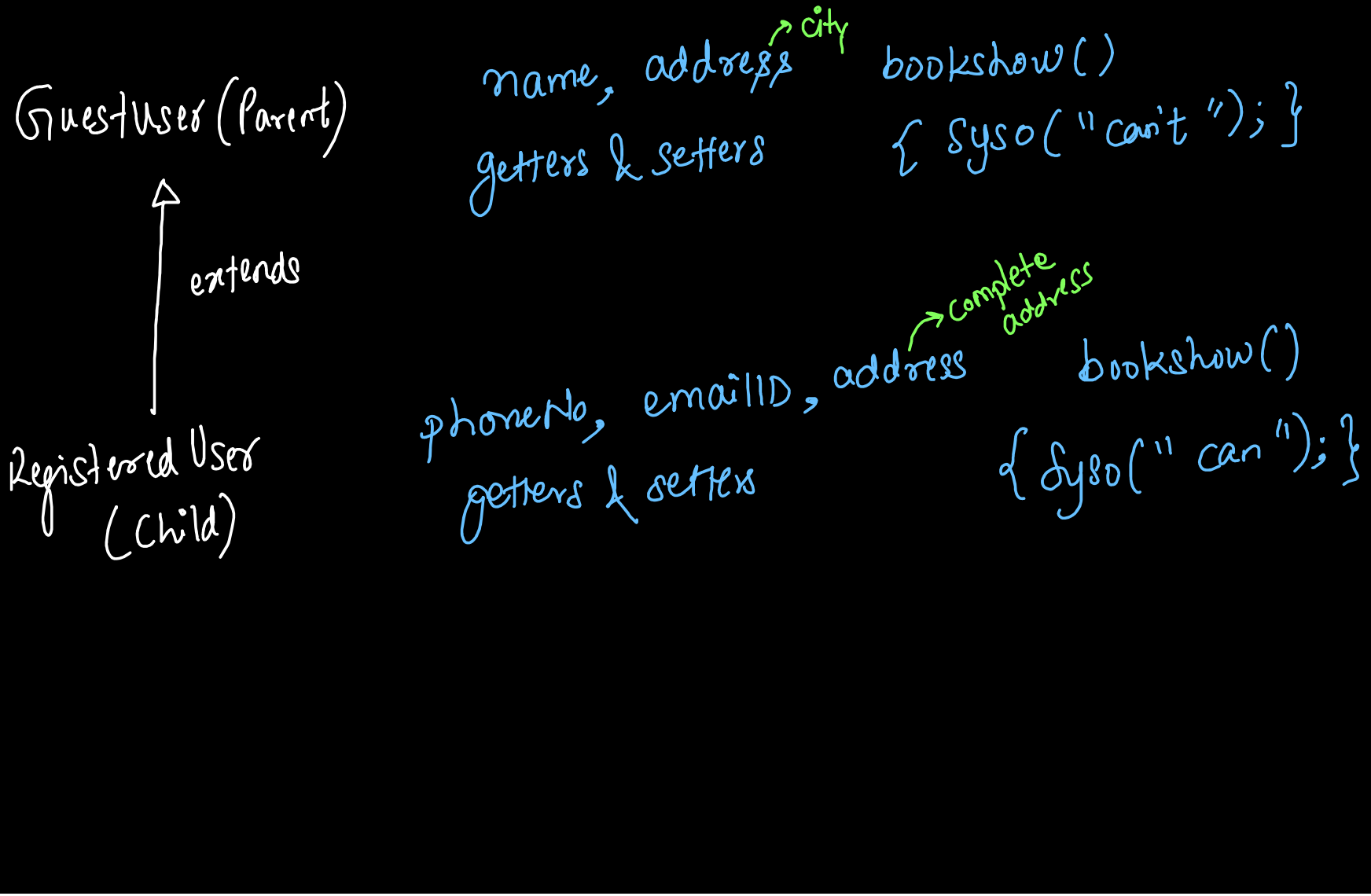


Q) What are Polymorphic Variables in Java? Give some real-world examples.

```java
class User {
    String name, address;

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public User(String name, String address) {
        this.name = name;
        this.name = address;
    }

    public void bookShow() {
        System.out.println(x: "You can't book show.
        Please register first");
    }
}
```

override →

```java
class RegisteredUser extends User {
    String phoneNo, emailId, address;

    public String getPhoneNo() {
        return phoneNo;
    }

    public void setPhoneNo(String phoneNo) {
        this.phoneNo = phoneNo;
    }

    public String getEmailId() {
        return emailId;
    }

    public void setEmailId(String emailId) {
        this.emailId = emailId;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }

    public RegisteredUser(String name, String address, String phoneNo, String
    emailId) {
        super(name, address: "Delhi");
        this.phoneNo = phoneNo;
        this.emailId = emailId;
        this.address = address;
    }

    public void bookShow() {
        System.out.println(x: "You can book the show, please proceed to payments");
    }
}
```


GuestUser (Parent)
extends
Registered User (Child)
name, address
city
getters & setters
bookshow()
{ syso("can't"); }
phoneNo, emailID, address
complete address
getters & setters
bookshow()
{ syso("can"); }

GuestUser (Parent)

name, address (city), bookshow()

getters & setters

{ syso("can't"); }

RegisteredUser (Child)

phoneNo, emailID, address (complete address), bookshow()

getters & setters

{ syso("can"); }

```
GuestUser user = new GuestUser();
```

reference variable

object/instance

```java
User u1 = new User(name: "Archit",
address: "Delhi");

System.out.println(u1.getName() + " " + u1.
getAddress());

u1.bookShow();

System.out.println(u1.name + " " + u1.
address);
```

```
Archit Delhi
You can't book show. Please register first
Archit Delhi
```

GuestUser (Parent)

name, address — accessible (indirectly) super

getters & setters

bookshow() — accessible (indirectly) super

{ Syso("can't"); }

extends

RegisteredUser (Child)

phoneNo, emailID, address

getters & setters

bookshow()

{ Syso("can"); }

user.address

child

parent

super.address

RegiUser user = new RegiUser();

reference variable

object/instance

```
RegisteredUser u2 = new RegisteredUser
(name: "Archit", address: "Delhi 110085",
phoneNo: "9319117889", emailId: "archit.
aggarwal023@gmail.com");

System.out.println(u2.name + ", " + u2.
emailId + ", " + u2.phoneNo);

u2.bookShow(); // overriding
System.out.println(u2.address);
System.out.println(u2.getAddress());
```

"can book the show"

this hides parent's address variable

```
Archit, archit.aggarwal023@gmail.com, 9319117889
You can book the show, please proceed to payments
Delhi 110085
Delhi
Delhi 110085
```

super.address (indirect)

this.address (direct access)

GuestUser (Parent)

extends

Registered User (Child)

name, address

getters & setters

overriden bookshow()

{ syso("can't"); }

@override

{created but not accessible}

phoneNo, emailID, address

getters & setters

overrided bookshow()

{ syso("can"); }

GuestUser User = new RegisteredUser();

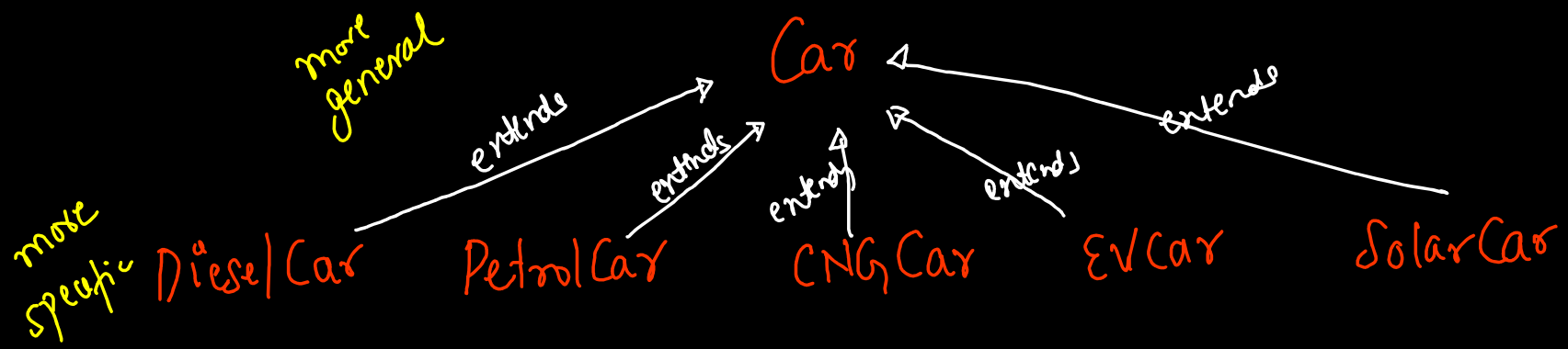
reference variable → variables & func^n prototype

Object/Instance → function body

Polymorphic Variables

⇒ Reference variables pointing to different objects are known as polymorphic variables.

↑ parents variable



```
DieselCar c1 = new DieselCar();
PetrolCar c2 = new PetrolCar();
CNGCar c3 = new CNGCar();
EVCar c4 = new EVCar();
SolarCar c5 = new SolarCar();
```

This is not preferred

```
Car c1 = new DieselCar();
Car c2 = new PetrolCar();
Car c3 = new CNGCar();
Car c4 = new EVCar();
Car c5 = new SolarCar();
```

Car is a polymorphic variable

RegisteredUser u4 = new GuestUser( );

emailId & phoneNo

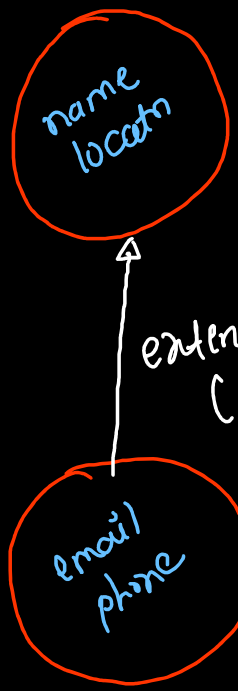
name, address

GuestUser() —child→ Subclass

RegUser() constructor

Q) What do you mean by upcasting and downcasting? Out of both, which is done implicitly and which needs to be done explicitly. Give some coding examples.

Q) What is instanceof operator and when to use it? What are the advantages of using it?



User u1 = new User(); ≅

RegUser u2 = new RegUser(); ≅ allowed

upcasting implicit

User u3 = new RegUser(); ≅ typecasting
Parent child

wrong { RegUser u4 = new User(); }
child Parent

RegUser u5 = u2; // shallow copy allowed

RegUser u6 = u3; // compilation error downcasting

RegUser u6 = (RegUser)u3; // allowed

RegUser u7 = u1; // compilation error

u7 = (RegisteredUser)u1; // runtime error

```java
public static void main(String[] args) {
    RegisteredUser u1 = new RegisteredUser(name: "archit", location: "delhi",
    emailId: "archit@gmail.com", phoneNo: 9319117888l);
    System.out.println(u1.name + " " + u1.emailId);

    // Child: Ref & Object

    User u2 = new User(); // Parent : Ref & Object
    System.out.println(u2.name);

    User u3 = new RegisteredUser(); // Parent Ref, Child Object
    System.out.println(u3.name + ((RegisteredUser) u3).emailId);

    // RegisteredUser u4 = new User(); child ref parent object not allowed

    RegisteredUser u5 = u1; // shallow copy
    System.out.println(u5.name + " " + u5.emailId);

    // RegisteredUser u6 = u3; // Compilation Error: No Typecasting
    if (u3 instanceof RegisteredUser) {
        RegisteredUser u6 = (RegisteredUser) u3;
        System.out.println(u6.name + u6.emailId);
    } else {
        System.out.println(x: "This user is not registered");
    }

    // RegisteredUser u7 = u2; // Compilation Error
    // RegisteredUser u7 = (RegisteredUser) u2;
    // Runtime Error: Classcast Exception

    if (u2 instanceof RegisteredUser) {
        RegisteredUser u7 = (RegisteredUser) u2;
        System.out.println(u7.emailId);
    } else {
        System.out.println(x: "This user is not registered");
    }
}
```

explicit down casting

```
archit archit@gmail.com
Guest User Created
Anonymous
Guest User Created
Registered User Created
AnonymousregisteredUser@gmail.com
archit archit@gmail.com
AnonymousregisteredUser@gmail.com
This user is not registered
```

Q) What do you mean by method hiding? Give some examples.

- overriding static methods
- overriding private methods
- change in parameters in overided methods

How is it different from method over-riding?

```
Parent{
    method()
}
```

⟹

```
Child extends Parent{
    method()
}
```

```java
class Parent {
    int parentData;

    public static void staticFun() {
        System.out.println(x: "This is parent's static function");
    }
}

class Child extends Parent {
    int childData;

    public static void staticFun() {
        System.out.println(x: "This is child's static function");
    }
}
```

```java
class Driver {
    @SuppressWarnings("all")
    Run | Debug
    public static void main(String[] args) {
        Parent obj1 = new Parent();
        obj1.staticFun();

        Child obj2 = new Child();
        obj2.staticFun();

        Parent obj3 = new Child();
        obj3.staticFun(); // Parent's Fun

        // No Dynamic Method Dispatch: No overriding
    }
}
```

Static Functions

Overriding ✗

Hiding ✓

dynamic binding ✗

static binding ✓

→ Parent

→ Child

→ Parent

```java
class Parent {
    int parentData;

    public static void staticFun() {
        System.out.println(x: "This is parent's static function");
    }

    public void privateFun() {
        System.out.println(x: "This is parent's private function
        but it is public");
    }

    public void publicFun() {
        System.out.println(x: "This is parents's fun with 0
        parameter");
    }
}
```

```java
class Child extends Parent {
    int childData;

    public static void staticFun() {
        System.out.println(x: "This is child's static function");
    }

    // private void privateFun() {}
    // Parent Public -> Child Private

    public void publicFun(int data) {
        System.out.println(x: "This is child's fun with 1
        parameter");
    }
}
```

```java
class Driver {
    @SuppressWarnings("all")
    Run | Debug
    public static void main(String[] args) {
        Parent obj1 = new Parent();
        obj1.staticFun();
        obj1.publicFun();
        // obj1.publicFun(10); // THis is not allowed

        Child obj2 = new Child();
        obj2.staticFun();

        obj2.publicFun();
        obj2.publicFun(data: 10);
        // Overloading with functions in different classes

        Parent obj3 = new Child();
        obj3.staticFun(); // Parent's Fun
        obj3.publicFun(); // Parent's Fun

        // No Dynamic Method Dispatch: No overriding
    }
}
```

Q) Why data members cannot be over-rided? What do you understand by variable hiding in Java?

Instance variable hiding refers to a state when instance variables of the same name are present in superclass and subclass.Now if we try to access using subclass object then instance variable of subclass hides instance variable of superclass irrespective of its return types.

Q) Can constructors be overrided? Give reasons.

No

accessible

super()

We can not override constructor as parent and child class can never have constructor with same name(Constructor name must always be same as Class name).

Q) Can overrided methods have different return types? What do you mean by covariant types?

The covariant return type specifies that the return type may vary in the same direction as the subclass. Before Java5, it was not possible to override any method by changing the return type. But now, since Java5, it is possible to override method by changing the return type if subclass overrides any method whose return type is Non-Primitive but it changes its return type to subclass type

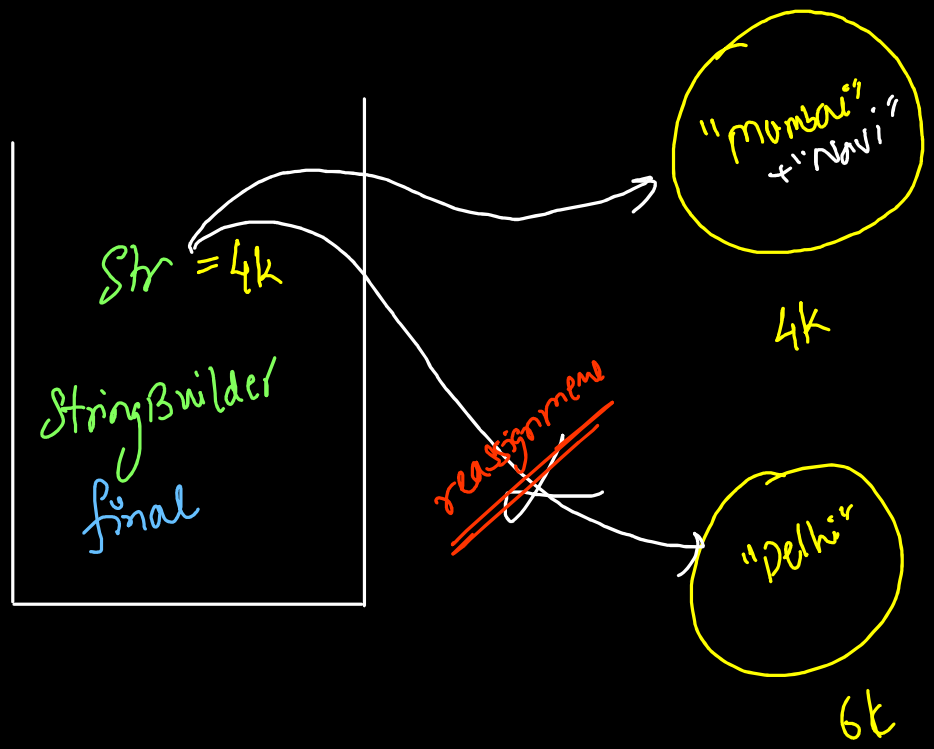
```java
class Parent {
    public void fun() {
        System.out.println(x: "This is parent's void fun");
    }

    public Parent getObject() {
        System.out.println(x: "This is parent's GetObject");
        return new Parent();
    }
}

class Child extends Parent {
    // public int fun(){
    // System.out.println("This is parent's int fun");
    // }
    // Invalid: Child should have void returntype

    @Override
    public Child getObject() {
        System.out.println(x: "This is child's GetObject");
        return new Child();
    }
    // Non-Primitive Covariant Type
}
```

```java
class Driver {
    Run | Debug
    public static void main(String[] args) {
        Parent obj1 = new Parent();
        Child obj2 = new Child();

        Parent obj3 = obj1.getObject();
        obj3.fun();
        // Parent Ref: Parent Object

        // Child obj4 = obj1.getObject(); // Not Valid

        Parent obj5 = obj2.getObject(); // Overriding
        obj5.fun();
        // Parent Ref: Child Object

        Child obj6 = obj2.getObject();
        obj6.fun();
        // Child Ref: Child Object
    }
}
```

```
This is parent's GetObject
This is parent's void fun
This is child's GetObject
This is parent's void fun
This is child's GetObject
This is parent's void fun
```

Q) What are the applications of final keyword?

- final primitive variables : constants → CAPITAL_LETTERS → Snake{underscore}
- final reference variables : Reference can't be changed but data(instance) can be.
- final methods : can't be over-riden (to stop runtime polymorphism) → Static Binding
- final class : can't be extended (to stop inheritance) // // inherited
  - A final class will automatically have all its functions final

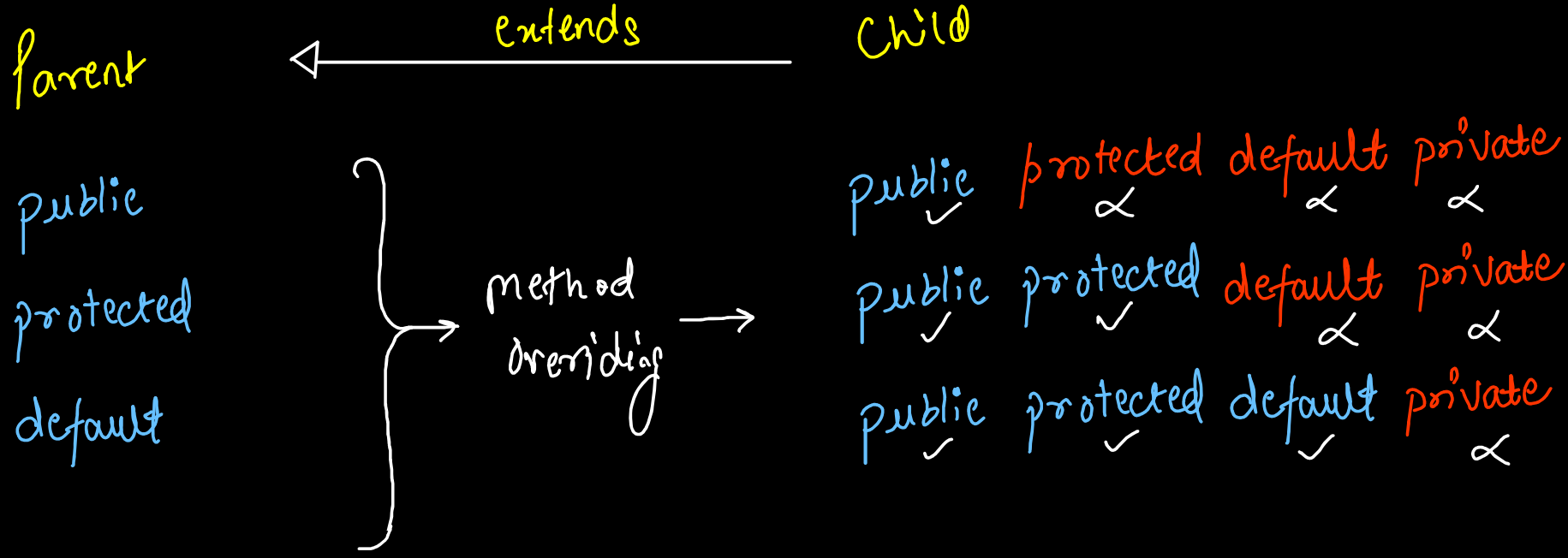


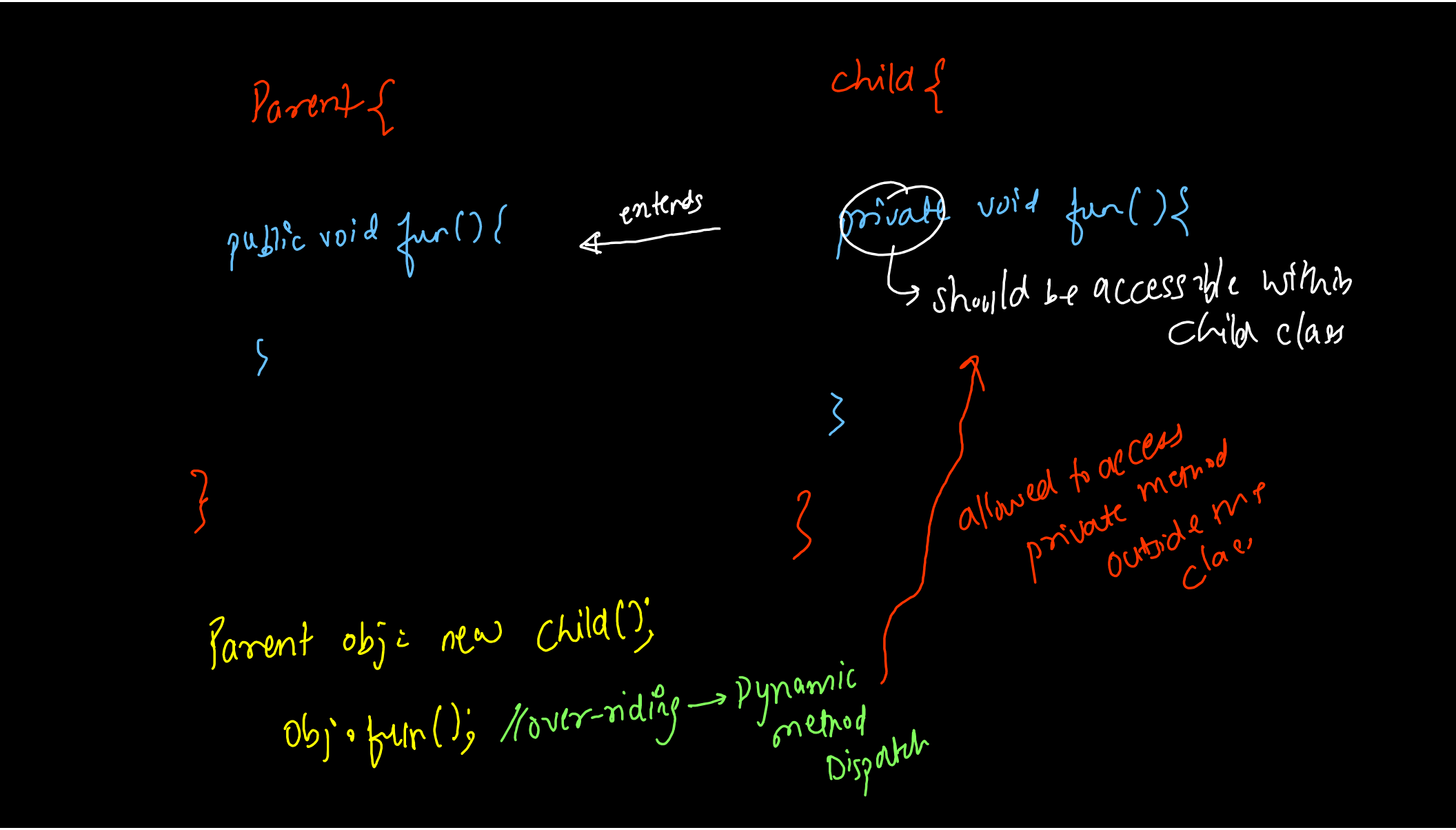

```java
class Driver {
    private static final double PI = 3.14;
    private static final StringBuilder str = new
    StringBuilder(str: "Mumbai");

    Run | Debug
    public static void main(String[] args) {
        // Get: Constant Variable: Allowed
        System.out.println(Driver.PI);

        // Set: Constant Variabel: Not Allowed
        // Driver.PI = 22/7.0;
        // Reassignment not possible

        System.out.println(Integer.MIN_VALUE);
        System.out.println(Integer.MAX_VALUE);

        // Get And Data Modification: Final Reference
        Variable
        System.out.println(str);
        str.append(str: " Navi");
        System.out.println(str);

        // Reassignment Not Allowed
        // str = new StringBuilder("Delhi");

    }
}
```

```java
class Parent {
    public final void finalFun() {
        System.out.println(x: "This is parent's fun");
    }
}

class Child extends Parent {
    // Final Method can't be overrided
    // public void finalFun(){}
}

final class Parent2 {
    public void finalFun() {
        System.out.println(x: "This is parent2's fun");
    }
}

// Final Class cannot be extended
// class Child2 extends Parent2{}
```

Access Modifiers & Method Overriding

Q) Explain the statement "Overrided method (child class) must be less restrictive than overriden method (parent class)"




Parent {
public void fun() {
}
}
extends
child {
private void fun() {
}
}
should be accessible within child class
Parent obj = new Child();
obj.fun(); //over-riding → Dynamic method Dispatch
allowed to access private method outside my class