(Part-10)

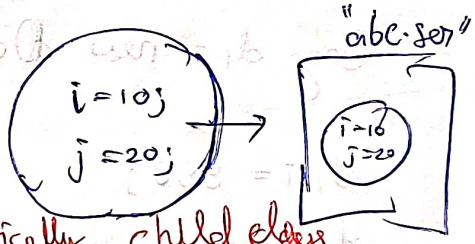
Concept of Serialisation introduced with Inheritance

(Case I): Parent Class Implements Serializeable, but child class does not implement Serializeable

```
class Animal implements Serializeable{
    int i=10;
}

class Dog extends Animal{
    int j=20;
}
```

```
Dog d1 = new Dog();
Serialize(d1);
```



Conclusion: If parent class is Serializeable, automatically child class becomes Serializeable.

In Java Generic Servelet Class is serializeable, all its child are as well.

(Part-11)

(Concept of Inheritance w.r.t Serialisation)

(Case II): Parent Class does not implement Serializeable, but child class implements Serializeable.

What happens then?

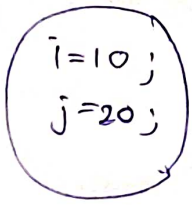
Lets see an Example to understand -

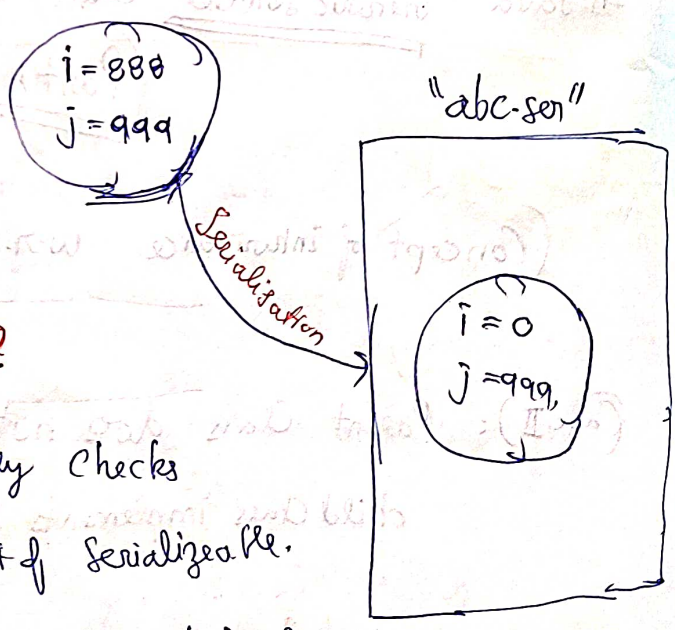
```
class Animal{
    int i=10;
}

class Dog extends Animal implements Serializeable{
    int j= 20;
}

psvm () throws Exception {
    Dog d1 = new Dog();
    d1.i = 888;
    d1.j= 999;

    Fos.....
    oos.....
    oos.writeObject (d1).
```





Q) What happened during Serialisation?

A) During Serialisation Jvm internally checks which instance variables are not part of Serializeable.

In our case i=10; → {From Animal which does not implement Serializeable}

So default value is stored for it i.e. 0.

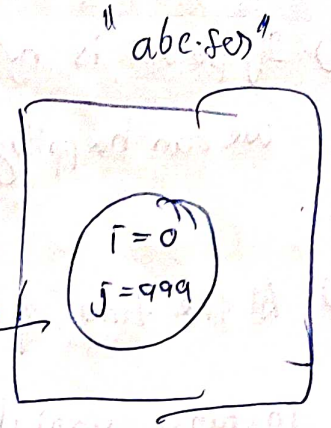
During "Deserialisation"

FIS ----

OIS ----

Dog d2 = (Dog) ois.readObject();

sout (d2.i + "----" + d2.j)



10 999

*** You see the above behaviour was not at all expected.

Why did (d2.i = 10) It should not happen.

During Deserialisation. JVM does 3 things.

1. Identification of instance members ⇒ Check which instance variables are not part of Serializable
2. Executions of instance variables, assignments & variables. ⇒ In Animal we saw i = 10.
3. Execution of default constructor. ⇒ JVM will call the No args default constructor of the parent or class which does not implement Serializable.

this is called (instance_control_flow)

JVM writes * instance_control_flow for classes which are not implementing Serializable

Rules.

① If parent is not Serializable yet the child is Serializable, we can happily Serialize the child class Object.

② At the time of Serialisation, JVM will check if any instance variable is coming (inheriting) from Non-Serializable parent. JVM ignores original value and saves default value to the file.

③ At the time of de-Serialisation, JVM will check if any parent is non-Serializable or not. Then JVM will execute instance control flow and share its instance variable values with current Object.

④ Inside any of the parent Classes which are non-Serializable, if we see they don't have a proper No args default constructor, it will throw RE: InvalidClass Exception → (Runtime Exception)

Important Code Execution

→ Code 1

```
class Animal{
    int i=10;
    Animal(){
        sout("Animal Constructor");
    }
}
```

If we remove this constructor and keep nothing, then JVM already has the default No args constructor.

```
class Dog extends Animal implements Serializable{
    int j=20;
    Dog(){
        sout("Dog constructor called");
    }
}
```

[Parent Class Constructor invoked before actual class]

```
psvm() throws Exception{
    Dog d1=new Dog();
    d1.i=888; d1.j=999;
    Serialize(d1);
    Dog d2=(Dog) Deserialize();
    sout(d2.i+----+d2.j);
}
```

When this is used First parent constructor is called.

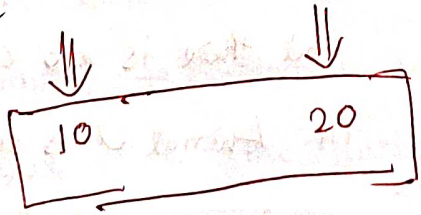
("Animal Constructor")

2nd Actual Class constructor is called

("Dog constructor called")

Instance-control-flow is invoked.

("Animal constructor")



(Code 2) {Imp: Example}

```
class Animal{
    int i=10;
    Animal(int i){
        this.i=i;
        sout("Animal Constructor");
    }
}
```

When this constructor is already written, Java will not implicitly add default constructor.

(**dd) Hence if user explicitly does not add default constructor then there will be none.

```
class Dog extends Animal implements Serializable{
    int j=20;
    Dog(){
        super(10);
        sout("Dog constructor");
    }
}
```

this we have to do else we will have Runtime Exception, As parent won't know which Constructor to invoke

```
psvm() throws Exception {
    Dog d1=new Dog();
    serialize(d1);
    Dog d2=(Dog) Deserialize();
    sout(d2.i+d2.j);
}
```

Dog's parent class constructor is called → ("Animal Constructor") ↓ ("Dog Constructor")

During Deserialization, @Instance_control_flow is invoked and there is no default constructor in Animal Class and,

RE: Invalid Class Exception