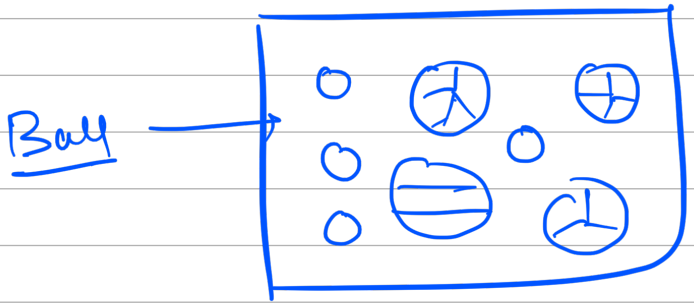
Agenda

Start @ 9:05

① Polymorphism
- Runtime time
- Compile time

Polymorphism = Many forms



There are multiple forms of Balls

```
Class A {
    Void print( )
    {
        System.out.print("Hello");
    }
    Void print(String s)
    {
        sout ("Hello" + s);
    }
}
```

print() method has different forms

```
psvm (){
    A a = new A();
    a.print();
    a.print("Scaler");
}
```

Method overloading

⇒ Because compiler decides which method to be called, it is called compile time polymorphism.

eg1 System.out.print (Integer)
(String)
(Object) ↓

eg2
```
Class Student {
    String Name;
    int age;
    double psp;
    String Batch;

    Student() {
    }
```

```
Student(String name)
{
    this.name = name;
}
```

Constructor Overloading

```
Student(String name, int age)
{
    this.name = name;
    this.age = age;
}
```

```
Create Student(String name, int age)
{
    if (name != null && age != null)
    → new Student(name, age)
    else if (name != null)
    → new Student(name)
    else
    → new Student();
}
```

Q1 ① Void print()
Void print(String s)

YES

② Void print(String s)
Void print(Integer i) YES

③ Void print(String s, int i)
Void print(int i, String s) YES

In Method Overloading order of parameters matter

④ Void print(String s)
String print(String s) NO

Method Signature

Void print(String s)

X ✓ ✓ X

Template

MethodName(Datatypes of parameter & its order)

Void print(String s, int i)

⇒ print(String, int)

void print(int i, String s)

⇒ print (int, String)

Formal Def^{n} of Method Overloading

⇒ Methods are known to be overloaded if they have same name but diff signature

Compile Time error

- void print(String s) ⇒ print(String)
- String print(String s) ⇒ print(String)

⇒ A class can't have two function with same signature

Doubt

```
Student {
    [int age=10;]
    Student( int age){
    }
    Student(){
    }
```

age=10 ←

Method Overriding (Runtime polymorphism)

```
Class A {
    void doSomething(String s) // doSomething(String)
    {
    }
}
```

Compile

```
Class B extends A {
    String doSomething(String s) // doSomething(String)
    {
    }
}
```

Overriding ⇒ Providing a new meaning to the existing function by overriding the implementation with out changing the function signature

eg

```
Class A {
    doSomething(String s)
    {
    }
}
```

```
Class B extends A
{
    doSomething (String s)
    {
    }
}
```

If parent and child have methods with same name, same signature and same return type then it is called Method Overriding

```
Class A {
    Void doSomething()
    {
        print ("Hello");
    }
}
```

```
A a = new B()
a.doSomething()

A a = new A();
```

```
Class B extends A {
    Void doSomething () {
    {
        print("Hi");
    }
}
```

PSVM()

① A a = new A()
a.doSomething() // hello

② a = new B()
a.doSomething() // hi

③ B b = new B();
b.doSomething(); // hi

④ B b = new A()
b.doSomething() // ERROR

String doSomething(String s)
Void doSomething(String s) — CTE

String doSomething(String s) → Parent
String doSomething(String s) → child

Only inheritance [RefVariable.doSomething()] → parent

B → A

String doSomething(String s) — A → B
String doSomething(Integer i) — B
String doSomething(Int i, String s) — B
↳ overloaded

Method overriding
1. Parent-child
2. Function sign should be same
3. Return type

```
Class A {
        void doSomething (String s)
}
Class B extend A
{       Void doSomething (String s)
        (int i)
{       ==========  {
```

Compile time Error

```
A a = new B()
```

Note The Method that will be triggered is decided by the object that has been created not by refernce.

⇒ Runtime Polymorphism (Dynamic Binding)

eg

```
Class A {
    print( ) {
        print("Hello");
    }
}
```

```
Class B extends A
{
    print( ) {
        print ("Hi");
    }
}
```

```
Class C extends A
```

```
List<A> users
```

{

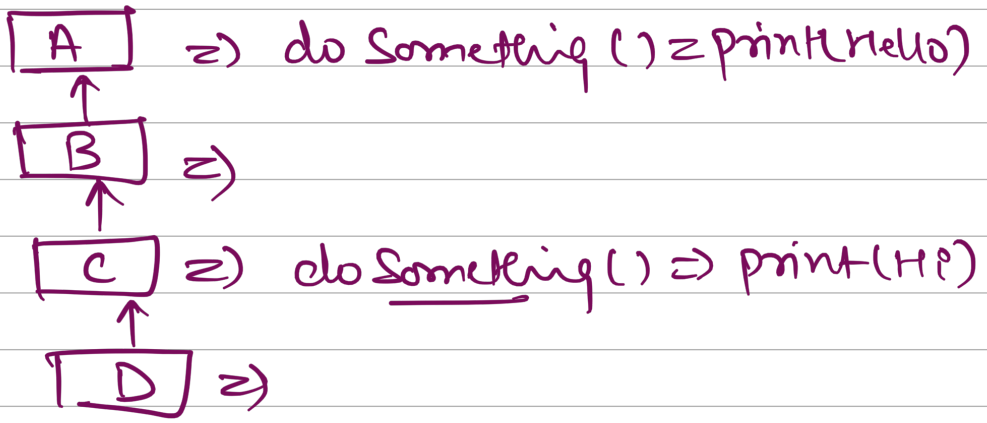
```
print() {
    print("Bye");
}
}
```

{ A(), B(), C(), A() } 4 elements

```
for(A a : users){
    a.print();
}
```

```
A a = new A();
A a = new B();
A a = new C();
```

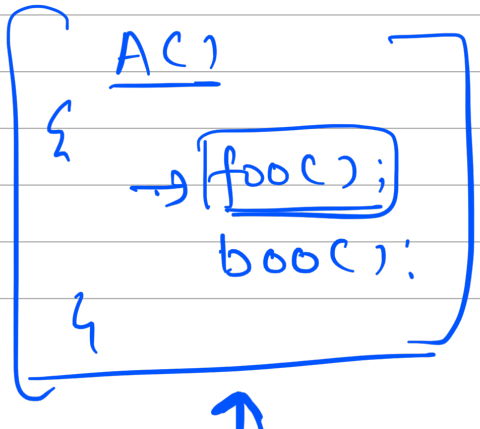
// hello, hi, bye, hello



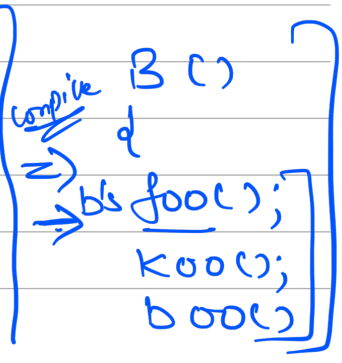

```
D d = new D();
d.doSomething() => Hi

A a = new D();
a.doSomething() => Hi

a = new B();
a.doSomething => hello
```



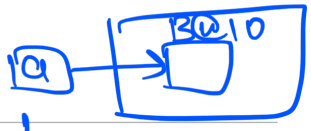

```
B extends A
{
    foo();
    koo();
}
```



ref

```
A a = new B();
a.foo();   ✓
a.boo();   ✓
a.koo();   X   → compile time error
```

Compile



B@10.foo();

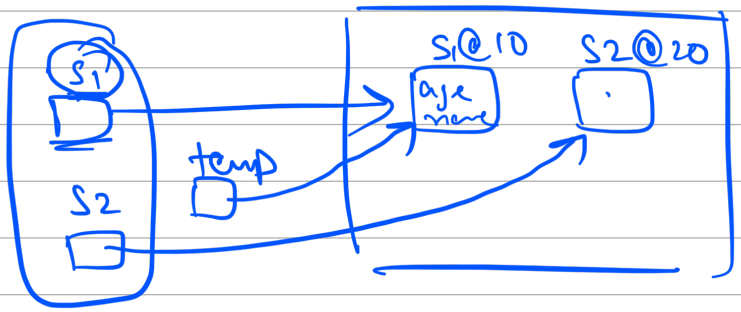
address of array

```
B b = (B) new A();   X Compile Error
```

Phonepe → BankApi → Bank1, Bank2

System.out.println(↔)

Student name
name + age
name + age + batch




```
S1 = new Student()
S2 = new Student()
```

swap(S1, S2)

function

```
temp = S1;   → S1.
S1 = S2;     → S2
S2 = temp    → S1.
```

Swap (Student S1, S2)

temp

{

}

S1@10 S2@2

Swap (Student S1, Student S2)

{

S1 = S2@20.

S2 = S1@10

}