Runtime Polymorphism
Early & Late Binding

(V. imp)

(Unit - 4)

~~Q) Why shud~~

- bp/ref → base ptr/ref.
- RTP → runtime polymorphism.
- CTP → Compile time poly.

Q) Why shud VF be called using base ptr/ref? Why not using obj?

A) Bcoz calling VF using bptr/ref allows us to achieve runtime poly.

Types of Poly.

- Compile Time (achieved using)
  - fun o/load
  - operator o/ld.
- Runtime (achieved using)
  - VF & bp/r

- In fn poly., there r mul fun with same name. Decision has to made as to wh. fun shud be called.

1) <u>CT Poly</u>

- In CTP, the decision is made @ ~~run-time~~ compile time.
- It is achieved by 2 ways

  1) <u>fun O/L :</u>

  - In this case, based on No./type of param, compiler will decide wh. fun to call..

eg There r three add fun().

```
add (int x)          { --- }
add (int x, int y)   { ---- }
add (int x, int y, int z)  { ---- }

int main()
{
    add (1);  -> I fun called
    add (1,2); -> II  "  "
    add (1,2,3); -> III  "  "
}
```

This is decided at compile time based on param.

2) operator O/L :

- Suppose there are 2 classes A & B.
- + operator is o/loaded for both classes
- So + operator can be used to add obj of A as well as obj of B.

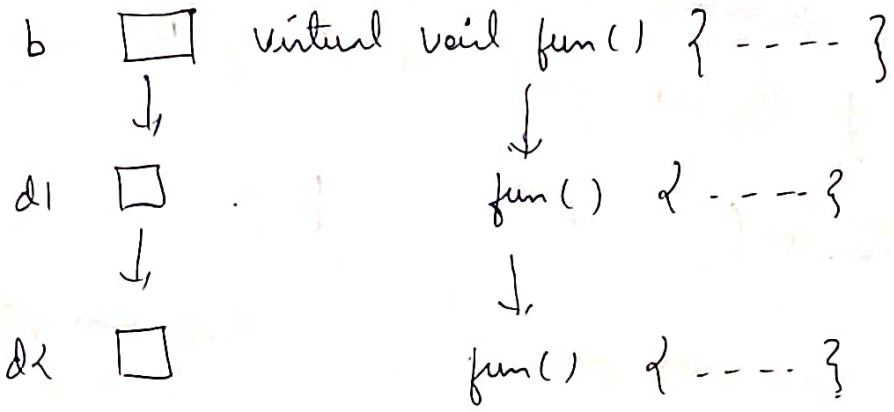
```
operator+ (.....)                    opr+ (----)
{                                    {
    // opr fun for                       //opr fun for
       class A                              class B
}                                    }
```

- obj-A1 + obj-A2;
  obj-B1 + obj-B2; // decided at compile time.

- If obj of A r added, then opr fun of class A will be called.
- If obj of B " " " " ...
  ... B ... " ... .
- cll. opr fun to call, this is decided at compile time. ∴ it is called CT poly.

Runtime Polymorphism

- RTP is achieved using VF & base ptr/ref. & fun o/R.



- Suppose each class has a fun()
- Now let's take 3 cases:-

1) b/ptr/ref pointing to base obj & calling fun() thru b/p/ref.

2) " " " " " d1 obj &
" " " " " " " " " " " "

3) " " " " " " " d2 obj &
" " " " " " " " " " " "

- In case-1: base fun will be called.
  Case-2: d1 . . . . . .
  3: d2 . . . . . .

- This decision is made during Runtime & not during compile time.
- ∴ it is called RTP.

eg
```
base  obj-b;
d1    obj-d1;
d2    obj-d2;
```
} Create obj of each class

```
base * bp;  //base ptr.
```

Case 1:
```
bp = &obj-b;
bp->fun();  // Base fun is called.
```

Case 2:
```
bp = &obj-d1;
bp -> fun();  // d1 fun is called.
```

Case 3:
```
bp = &obj-d2;
bp->fun();  // d2 fun is called.
```

} Which fun to call, this decision is made at run time

Example of a RTP

- RTP allows you to pass diff obj to the same fun.
- Create a VF in base class & ~~over~~ o/ride in der classes.
- Then create 2 fun
  In example1() base obj is passed as param.
  " example2() base ref " " " ....
- These example() fun will simply call the VF.

- Inside main(), 3 obj r created - base, d1, d2
- In example 1(), only base-obj can be passed. d1 & d2 obj can't be passed. So this is an example of CTP.

- In example 2(), all types of obj can be passed. So it is an example of RTP.

- Therefore RTP allows you to pass any type of obj to a fun.

* This is the adv of RTP - it allows more flexibility **.

eg
```
Class base
{
        virtual void fun()
          { " base " }

};
Class d1 : public base
{
        void fun()
          { " d1 " }

};
Class d2 : public d1
{
        void fun()
          { " d2 " }

};
```

```
void example1(base obj_b)
{
    obj_b.fun();
    //simply call the
}       VF inside example fun.
```

→ base obj is used as a param.

```
~~void explame~~
void example2(base &ref)
{
    ~~obj~~ ref.fun();
    // call VF inside this

}

int main()
{:  base obj_b;
    d1 obj_d1;
    d2 obj_d2;

    example1(obj_b);
X   example1(obj_d1); X
X   example1(obj_d2); X

example2(obj_d);
example2(obj_d1);
example2(obj_d2);

}
```

→ only base obj can be passed in example1 bcoz it is using a base_obj as param

//This is CTP.

All types of obj can be passed in eg2 bcoz it is using bref as param.

//This is RTP.

//RTP provides flexibility - any obj can be passed to example2.

** imp

CTP	RTP
1) In CTP, the call is resolved by compiler	1) Not resolved by compiler
2) also known as static binding, early binding & O/L	2) Dy. binding, late " overriding
3) Fun O/L is used	3) Fun O/R is used.
4) Achieved thru F/O/L & opr O/L	4) achieved thru VF & b. p/r
5) It is faster	5) Slower.
6) Less flexible	6) More flexible

Early & Late Binding

Q What is binding?

A • When a fun is called, then the compiler will bind the fun call with fun definition

```
void add();
{
  ==
}
```

```
add();
```

• This is also called as resolving the fun call.

Q What is early binding & late binding?

A Early Bind^g

• In CTP, the fun call is binded/resolved at compile time. This is called EB.

• It is done during normal fun call, fun o/L or opr o/L.

Late Bind^g

• In RTP, the fun call is resolved at RT.

• It is done when calling a VF thru a b/p/r.

* imp

EB	LB
1) Class info is used to resolve fun call	1) obj info is used to resolve fun call.
2) Occurs at Compile T	2) RT.
3) Static binding	3) Dy.
4) fun o/L, opr o/L	4) fun o/R, VF.
5) No use of bp/r & VF	5) bp/r r used. & VF.
6) Fast	6) Slow.
7) ~~Flexible~~ Not flexible	7) ~~Not~~ Flexible.