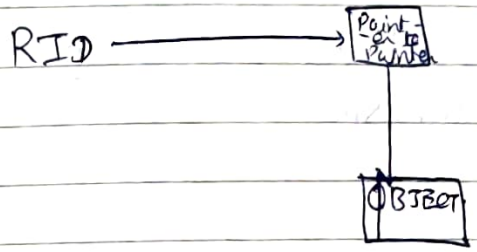
Ref. ID



* ~~We~~ RID dont direcly point to address of Object, So It ~~per~~ uses pointer to point-er for Security Reasons.

POLYMORPHISM :—

Whenever an Object gives a different-diff-rent in a different-different circumstan-ces, this behaviour of Object is called Polymorphis.

eg:- like an AC can give 'hot' & 'cold' both air at diff. setting.

Advantage

Polymorphis reduces the complexity of program.

Eg

```
System.out.println("Hello");
                  (1.10);
                  (2.4);
                  ('a');
```

println(String s);
println(int x);
→ only one println is used to call all of Data types

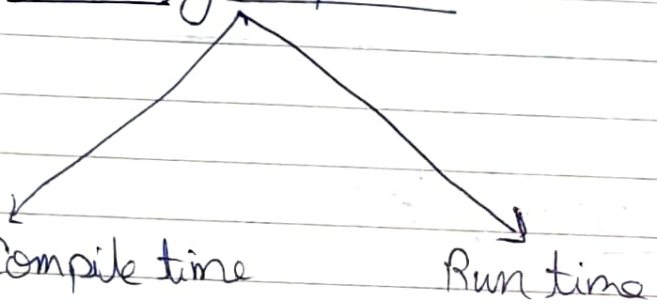
RULE

1. Polymorphism is always achived via behaviour of an object.

2. Properties of an object don't play role in case of Polymorphism.

23/07/2017

Two tyes of Polymorphism.



Compile time Polymorphism

Whenever an object is bound with their f^nality at compile time, this is known as C.T. polymorphis but object is not available at C.T.

Runtime Polymorphism

Whenever an object is bound with theis f^nality at Run time.

Date / /
Page

For OOPS

Compile Time
- F^n Overloading.
- Operator Overloading

Runtime
- F^n Overriding.

Changes made by Java.

(i) Java doesn't support Operator overloading explicitly. But Supports implicitly.

Eg of Polymorphism via '+' operator

$20 + 30 = 50$

"good" + "morning" = "good morning"

Here '+' is overloaded. because '+' is performing many functionality.

RULE

- Java doesn't support compile time polymorphis.
- In Java made in 'F^n Overloading' case, binding occurs in 'Runtime polymorphis'.
- In other OOPS ~~super~~ based lang. there are virtual f^n

- In Java by default, all the instance f^n's are implicit virtual.
- But in Static f^n, the binding happens in Compile Time (Only case of C.T.B. in Java) But binding is to 'class' not 'Object'. The 'Static' f^n is only present in Java. So, its not a concept of OOPS. (→ concept of Static f^n)

Method Invocation

- The Java prog. ing lang. provides two basic methods
  (i) Instance :-
  (ii) Class (or Static)

1. Instance methods use dynamic (late) binding; Class method use static (early) binding.

3. The JVM uses 2 diff. instruc^n to invoke these 2 diff. kinds of Methods.
   (i) 'invokevirtual' for instance methods.
   (ii) 'invokestatic' for class methods.

Date / /
Page

F^n Overloading

" " says you can have more than 1 f^n with a same name in a one class, having a diff. prototype.

(1) Access Specifier (2) Access Modifier.

(3) Return Type (4) function name (Argument) (5)

public static void main(String S[])
① ② ③ ④ ⑤

- But Java ~~or~~ didn't ~~use~~ have concept 'Specifier' it made all of them 'Modifier'.

Modifiers

(i) public
(ii) protected.
(iii) private.
(iv) default
(v) static
(vi) final
(vii) abstract
(viii) volatile
(ix).

Date / /
Page

RULE

(1.) Access Modifiers don't play any role in case of F^n overloading.

public void show() } Not
private " show() } F^n over-
static " " } -loading

$\Rightarrow e_1.show$
↳ will not differentiate.

2. F^n Name can't be changed for F^n overloading.

Achiving F^n Overloading by change in ARGUMENTS

1. Change the No. of arguments in each f^n.

eg

show(10) → void show() ✗
void show(int x) ✓
" " (int x, int y) ✓
" " (" , " , char c) ✓

Date / /
Page

26/06/17

2. If you want to keep, the no. of arguments in each fn same, & still want to overload them, then change the datatype of ~~the~~ their arguments

Eg.

show (10) ⟶
void show(byte b)
void show(short s)
void show(int x)
void show(long l)

Now

void show(int x, long l)
void show(long l, int x)
} Will compile without no problem as data-type are diff

(1) show (10, 10L) → (i)
(2) show (10L, 10) → (ii)
(iii) show (10, 10) → compilation ~~error~~ error
becoz of exact match to both fn after promotion, ~~because~~ hence gives ambiguity ~~error~~ error.

Achiving a Fn Overloading by changing only in 'Return' Type

Eg

```
Void show (int x)
 {

 }

int    show(int x)
 {
  return x;
 }
```

* Its Not Mendetory to catch the return value of any fn while calling it.

```
g.
show(10);
```

will go without any error

```
int show(int x)
{
return x;
}
```

* All compilers search fn only on name of fn. & arguments. Not not return type.

Eg - 1

int x = show(10)
or
show(3c)

Will both fail give error

```
void show(int x)
{
}

int show(int x)
{
  return 20;
}
```

Eg - 2

show(10) ← (i) void show(int x)
{
}

show('Acc') ← (i) int show(char x)
{
return 20
}.

RULE

Return Type of fns don't play any role in case of fn overloading.

Fn overloading can only be achieved by change only in Argument.