Object Oriented Programming
CS F213

J. Jennifer Ranjani

email: jennifer.ranjani@pilani.bits-pilani.ac.in

Chamber: 6121 P, NAB

Consultation: Friday 4.00 – 5.00 p.m.

BITS Pilani

Pilani Campus

Method & Constructor Overloading

BITS Pilani
Pilani Campus

Method Overloading

- Multiple methods having same name but different parameters is known as method overloading.

- Eg. Suppose you have to perform addition of the given numbers but there can be any number of arguments, if you write the method such as a(int,int) for two parameters, and b(int,int,int) for three parameters then it may be difficult for you as well as other programmers to understand the behavior of the method because its name differs.

- Adv: increases the readability of the program.

Different ways to overload

- Changing the number of arguments
  - **int** add(**int** a,**int** b)
  - **int** add(**int** a,**int** b,**int** c)
- Changing the data type
  - **int** add(**int** a, **int** b)
  - **double** add(**double** a, **double** b)
- Changing only the return type does not mean method overloading
  - **int** add(**int** a,**int** b)
  - **double** add(**int** a,**int** b)
  - Compile Time Error: method add(int,int) is already defined in class Adder

Method Overloading - Example

innovate achieve lead

```
class Account{
int acc_no;
String name;
float amount;
void insert(int a,String n,float amt){
        acc_no=a;
        name=n;
        amount=amt;  }
void insert(int a,String n){
        acc_no=a;
        name=n;
        amount=1000;  }
void display(){
        System.out.println(acc_no+" "+name+" "+amount);}
}
```

Minimum balance is 1000

Method Overloading - Example

innovate achieve lead

```
class TestAccount{
public static void main(String[] args){
        Account a1 =  new Account();
        a1.insert(832345,"Ankit",5000);
        a1.display();

        Account a2 =  new Account();
        a2.insert(832346,“Shobit“);
        a2.display();
}}
```

Output:
832345 Ankit 5000.0
832346 Shobit 1000.0

Can Main() be overloaded?

innovate achieve lead

```
public static void main(String[] args){System.out.println("main with String[]");}
public static void main(String args){System.out.println("main with String");}
public static void main(){System.out.println("main without args");}
```

Ans: Yes. JVM calls main() method which receives **string array** as arguments only.

Overloading and Type Promotion

innovate achieve lead

```
class OverloadingCalculation{
  void add(int a,long b){System.out.println(a+b);}
  void add(int a,int b,int c){System.out.println(a+b+c);}

  public static void main(String args[]){
  OverloadingCalculation obj=new OverloadingCalculation();
  obj.add(20,20);
  obj.add(20,20,20);
  }
}
```

Output:
40
60

Overloading and Type Promotion (Matching Type Arguments)


```
class OverloadingCalculation{
  void add(int a,int b){System.out.println("int arg method invoked");}
  void add(long a,long b){System.out.println("long arg method invoked");}


  public static void main(String args[]){
  OverloadingCalculation obj=new OverloadingCalculation();
  obj.add(20,20);
  }
}
```

Output:
int arg method invoked

Overloading and Type Promotion (Ambiguity)

innovate achieve lead

```
class OverloadingCalculation{
  void add(int a,long b){System.out.println("a method invoked");}
  void add(long a,int b){System.out.println("b method invoked");}

  public static void main(String args[]){
  OverloadingCalculation obj=new OverloadingCalculation();
  obj.add(20,20);

  }
}
```

Output:
Compile time error

Constructor Overloading

- **Recall: Constructor is just like a method but without return type.**
- **Constructor overloading:** Having more than one constructor with different parameter lists.
- The compiler differentiates by the **number of parameters** in the list and their **types**.

Constructor Overloading-Example

innovate achieve lead

```
class Account{
int acc_no;
String name;
float amount;
Account(int acc,String aname){
acc_no = acc;
name = aname;
amount = 1000;   }
Account(int acc,String aname, float amt){
acc_no = acc;
name = aname;
amount = amt;  }
void display(){
System.out.println(acc_no+" "+name+" "+amount);}
}
```

Constructor Overloading-Example


```
class TestAccount{
public static void main(String[] args){
Account a1=new Account(832345,"Ankit",5000);
a1.display();
Account a2=new Account(832346,”Shobit");
a2.display();
 }}
```

Output:
832345 Ankit 5000.0
832346 Shobit 1000.0

Passing Objects to Constructors-Example

```
class Account{
int acc;
String name;
float amount;
Account(int act,String aname){
acc = act;
name = aname;  }
Account(Account a){
acc = a.acc;
name = a.name;   }
boolean equalTo(Account a) {
return(acc == a.acc && name == a.name);  }
void display(){
System.out.println(acc+" "+name+" "+amount);}
}
```

Passing Objects to Constructors-Example

innovate achieve lead

```
class TestAccount{
public static void main(String[] args){
Account a1=new Account(832345,"Ankit");
Account a2 = new Account(a1);
Account a3=new Account(832346,"Shobit");

System.out.println("a1==a2: " + a2.equalTo(a1));
System.out.println("a1==a3: " + a3.equalTo(a1));

a1.name="Aankit";
a1.display();
a2.display();

}}
```

Output:

a1==a2: true
a1==a3: false
832345 Aankit 0.0
832345 Ankit 0.0

innovate achieve lead

BITS Pilani

Pilani Campus

‘This’ Keyword

‘this’ Keyword

- It is a reference variable that refers to the current object

- Six usage
  - this can be used to refer current class instance variable.
  - this can be used to invoke current class method (implicitly)
  - this() can be used to invoke current class constructor.
  - this can be passed as an argument in the method call.
  - this can be passed as argument in the constructor call.
  - this can be used to return the current class instance from the method.

this: to refer current class instance variable


```
class Account{
int acc;
String name;
float amount;
Account(int acc,String name, float amount){
acc = acc;
name = name;
amount = amount;  }
void display(){
System.out.println(acc+" "+name+" "+amount);}
}
class TestAccount{
public static void main(String[] args){
Account a1=new Account(832345,"Ankit",5000);
a1.display();
 }}
```

Name of instance variables and formal arguments are same

Output:
0 null 0.0

this: to refer current class instance variable

```
class Account{
int acc;
String name;
float amount;
Account(int acc,String name, float amount){
this.acc = acc;
this.name = name;
this.amount = amount;  }
void display(){
System.out.println(acc+" "+name+" "+amount);}
}
class TestAccount{
public static void main(String[] args){
Account a1=new Account(832345,"Ankit",5000);
a1.display();
 }}
```

Output:
832345 Ankit 5000

this: to invoke current class method

innovate achieve lead

```
class Account{
int acc;
String name;
float amount;
void insert(int acc,String name, float amount){
this.acc = acc;
this.name = name;
this.amount = amount;
this.display(); }
void display(){
System.out.println(acc+" "+name+" "+amount);}
}
class TestAccount{
public static void main(String[] args){
Account a1=new Account();
a1.insert(832345,"Ankit",5000);  }}
```

If the function is invoked as display(), the compiler automatically adds this keyword