BITS Pilani
Pilani Campus

Object Oriented Programming CS F213

J. Jennifer Ranjani
email: jennifer.ranjani@pilani.bits-pilani.ac.in
Chamber: 6121 B, NAB
Consultation: Appointment by e-mail

Java Objects

BITS Pilani
Pilani Campus

Passing Objects to Methods

- Java is strictly pass by value
- Call by reference can be achieved when objects are passed as arguments
  - When a variable of class type is created, it implies that a reference to an object is created.
    - Eg: Account a1;
  - Reference variable is used to store the address of the object.
  - When the reference is passed to a method, the parameter that receives refer to the same object.

Passing Objects - Example

```
class Account{
int acc;
String name;
float amount;

Account(int act,String aname){
acc = act;
name = aname;
}

boolean equalTo(Account a) {
return(acc == a.acc && name == a.name);
}


}
```

Passing Objects - Example

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

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

}}
```

Output:

```
a1==a2: true
a1==a3: false
```

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

```
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
```

Assigning Object Reference Variables

- Value of a reference variable can to assigned to another reference variable.

- Assigning reference will not create distinct copies of objects.

- All reference variables are referring to the same object.

Assigning Object Reference

```
class Account{
int acc;
String name;
float amount;
Account(int act,String aname){
acc = act;
name = aname;
}

boolean equalTo(Account a) {
return(acc == a.acc && name == a.name);
}
void display(){
System.out.println(acc+" "+name+" "+amount);}
}
```

Assigning Object Reference

```
class second{
public static void main(String[] args){
Account a1=new Account(832345,"Ankit");
Account a2= 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 Aankit 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

```
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

this() : to invoke current class constructor

- The this() constructor call can be used to invoke the current class constructor. It is used to reuse the constructor. In other words, it is used for constructor chaining.

- Calling default constructor from parameterized constructor

- Calling parameterized constructor from default constructor

Constructor Chaining - Example

```
class Account{
int acc;
String name;
float amount;
Account(int acc, String name){
this.acc = acc;
this.name = name;}

Account(int acc, String name, float amount){
this.acc = acc;
this.name = name;
this.amount = amount;  }

void display(){
System.out.println(acc+" "+name+" "+amount);}
}
```

Constructor Chaining - Example

```
class Account{
int acc;
String name;
float amount;
Account(int acc, String name){
this.acc = acc;
this.name = name;}

Account(int acc, String name, float amount){
this(acc, name);  //reusing constructor
this.amount = amount;  }

void display(){
System.out.println(acc+" "+name+" "+amount);}
}
```

Constructor Chaining - Example

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

~~Account(int acc, String name, float amount){~~
~~this.amount = amount;~~
~~this(acc, name); //reusing constructor }~~

this: to pass as an argument in the method

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

this: to pass as an argument in the method

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

}}
```

Output:

```
832345 Shobit 0.0
832346 Aankit 5000.0
832346 Aankit 5000.0
```

this: to pass as argument in the constructor call

```
class Branch{
 Account obj;
int branch;
 Branch(Account obj){
   this.obj=obj;
   this.branch = 111;
 }
 void display(){
   System.out.println(this.obj.acc+" "+this.obj.name+" "+this.branch);
 }
}
```

this: to pass as argument in the constructor call

```
class Account{
  int acc;
  String name;
  Account(int acc,String name){
   this.acc=acc;
   this.name =name;
   Branch b=new Branch(this);
   b.display();
  }
}

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

Output:

```
832345 Ankit 111
```

Returning Objects using this keyword

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

Returning Objects using this keyword

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

innovate achieve lead

BITS Pilani
Pilani Campus

Mutable and Immutable Objects

Immutability and Instances

- Mutable Objects: Contents of an instance that can be modified.

- Eg: Immutable: java.lang.String

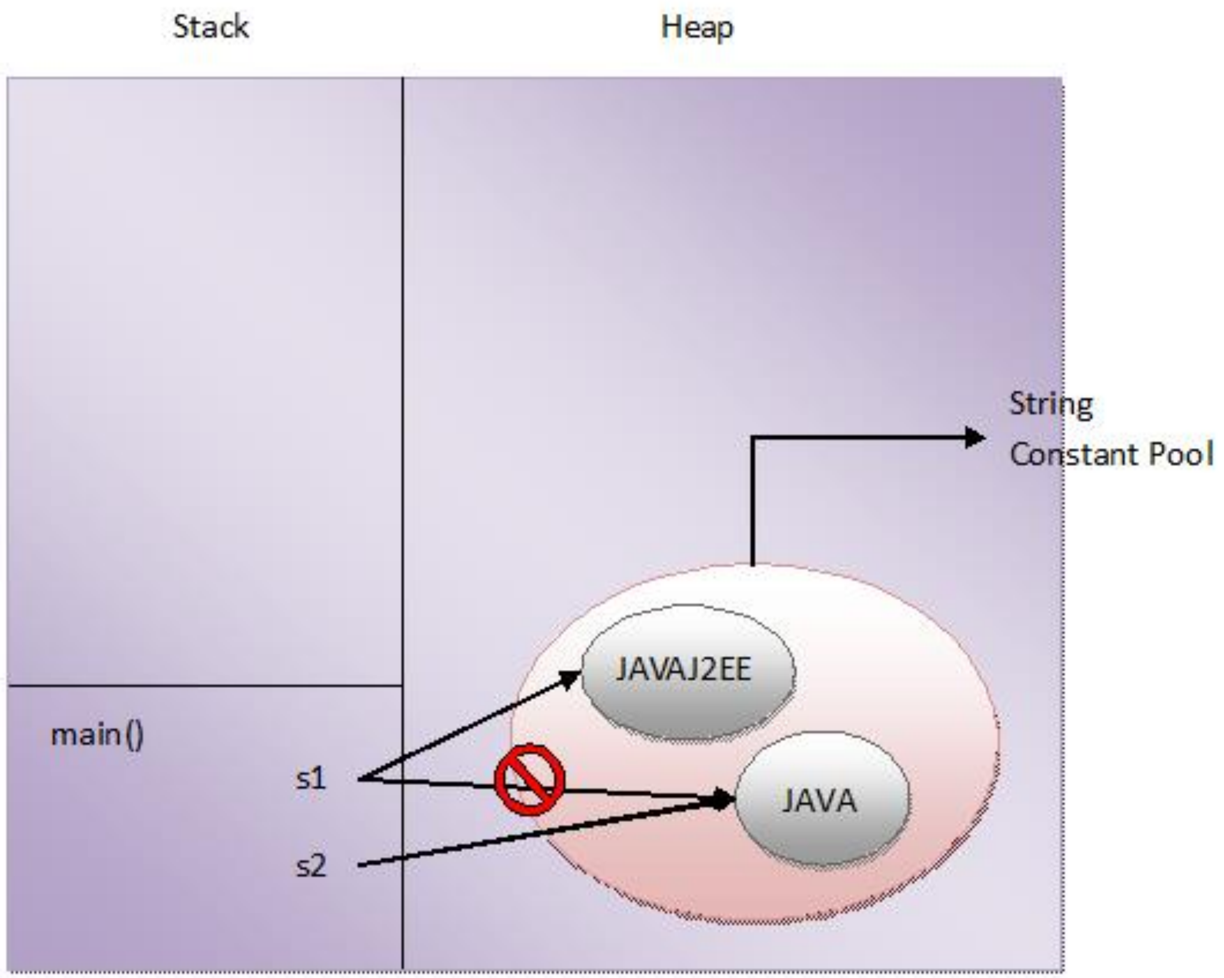
  Mutable: Account

- When the contents of the String instance are modified, a new string object is created.

Example

```
public class StringExamples
{
    public static void main(String[] args)
    {
        String s1 = "JAVA";

        String s2 = "JAVA";

        System.out.println(s1 == s2);         //Output : true

        s1 = s1 + "J2EE";

        System.out.println(s1 == s2);         //Output : false
    }
}
```

What happens in memory?



Are Strings created using new operator also immutable?

```
public class StringExamples
{
    public static void main(String[] args)
    {
        String s1 = new String("JAVA");

        System.out.println(s1);        //Output : JAVA

        s1.concat("J2EE");

        System.out.println(s1);        //Output : JAVA
    }
}
```

Are Strings created using new operator also immutable?

```
public class StringExamples
{
    public static void main(String[] args)
    {
        String s1 = new String("JAVA");

        System.out.println(s1);          //Output : JAVA

        String s2=s1.concat("J2EE");

        System.out.println("s1: "+s1+" s2: "+s2);          //Output : s1: JAVA s2: JAVAJ2EE

     }
}
```

How to create an Immutable class?

- Class must be declared as final
  - So that child classes can’t be created
- Data members in the class must be declared as final
  - So that we can’t change the value of it after object creation
- A parameterized constructor
- Getter method for all the variables in it
- No setters
  - To not have option to change the value of the instance variable

Immutable Class - Example

```
final class Account{
final int acc;
final String name;
final float amount;

Account(int acc,String name,float amt){
this.acc = acc;
this.name = name;
this.amount = amt;   }

int getAcc(){
return acc;}
String getName() {
return name; }
float getAmount() {
return amount;  }}
```

Immutable Class - Example

```
class TestAccount{
public static void main(String[] args) {

Account a= new Account(111,"Ankit",5000);

System.out.println("Acc: "+a.getAcc()+" Name: "+a.name);

a.amount = 1000;
}}
```

Output:

```
Exception in thread "main" java.lang.Error:
Unresolved compilation problem:
The final field Account.amount cannot be
assigned
```