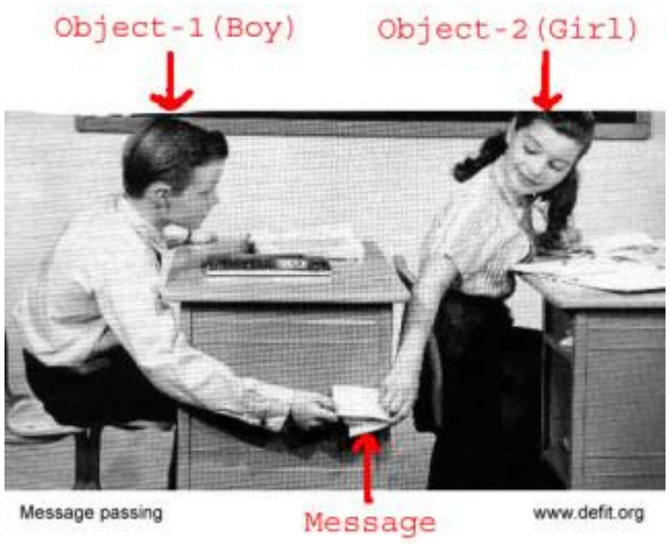
LEC4 OOP 2018-2019

Message passing

Objects communicate with one another by sending *messages*. A message is a *method call* from a message-sending object to a message-receiving object. An object responds to a message by executing one of its methods. Additional information, known as *arguments*, may accompany a method call. Such parameterization allows for added flexibility in message passing. The set of methods collectively defines the dynamic behavior of an object. An object may have as many methods as required.



Message Components

A message is composed of three components:

• an object identifier that indicates the message receiver,
• a method name (corresponding to a method of the receiver), and
• arguments (additional information required for the execution of the method).

What exactly does the statement mean in code?

```
class A { // starting of class definition
   methodA()    {    } // this is a class body (definition)
} // Ending of class definition

class B {
   methodB()   {    }// this is a class body (definition)
}
class C {
   main()    {
      A a=new A();        B b=new B();  // a is an object and  A is an abstract data type ADT
      a.methodA();        b.methodB();     } }// message passing
```

Polymorphism -Method Overloading-

Method overloading is a feature that allows a class to have more than one **method** having the same name, if their argument lists are different.

<u>Three ways to overload a method:</u>

In order to overload a method, the argument lists of the methods must differ in either of these:

This is a valid case of overloading

1. Number of parameters.

For example:

```
public int add(int, int) {----------}
public int add(int, int, int) {------}
```

2. Data type of parameters.

For example:

```
public int add(int, int) {-----}
public int add(int, float){-----}
```

3. Sequence of Data type of parameters.

For example:

```
public int add(int, float) {-----}
public int add(float, int) {-----}
```

4. Number and data type of parameters

```
public int  add(int, int) {-----}
public int  add(int, float, int) {-----}
```

Invalid case of method overloading:

When I say argument list**<u>, I am not talking about return type of the method</u>**, for example if two methods have same name, same parameters and have different return type, then this is not a valid method overloading example. This will throw compilation error.

```
public int add(int, int) {-----}

public float add(int, int) {-----}
```

LEC4 OOP 2018-2019

Method overloading is an example of ***Static Polymorphism***. We will discuss polymorphism and types of it in a NEXT LECTURES

Points to Note (ملاحظات)

1. Static Polymorphism is also known as compile time binding or early binding.
2. Static binding happens at compile time. Method overloading is an example of static binding where binding of method call to its definition happens at Compile time.

Trace (تتبع) the following examples:

Example 1: Overloading – Different Number of parameters in argument list

This example shows how method overloading is done by having different number of parameters

```
class DisplayOverloading {
   public void disp(char c)     {
        System.out.println(c);     }
   public void disp(char c, int num)      {
        System.out.println(c + " "+num);    }
}
class Sample {
  public static void main(String args[])    {
      DisplayOverloading obj = new DisplayOverloading();
      obj.disp('a');
      obj.disp('a',10);    } }
```

Output:

```
a
a 10
```

Explanation (التوضيح)

In the above example – method disp() is overloaded based on the number of parameters – We have two methods with the name disp but the parameters they have are different. Both are having different number of parameters.

Example 2: Overloading – Difference in data type of parameters

In this example, method disp() is overloaded based on the data type of parameters – We have two methods with the name disp(), one with parameter of char type and another method with the parameter of int type.

```
class DisplayOverloading2 {
   public void disp(char c)     {
       System.out.println(c);     }
   public void disp(int c)     {
      System.out.println(c );    } }
```

```
class Sample2 {
   public static void main(String args[])     {
      DisplayOverloading2 obj = new DisplayOverloading2();
      obj.disp('a');
      obj.disp(5);     } }
```

Output:

```
a
5
```

Example3: Overloading – Sequence of data type of arguments

Here method disp() is overloaded based on sequence of data type of parameters – Both the methods have different sequence of data type in argument list. First method is having argument list as (char, int) and second is having (int, char). Since the sequence is different, the method can be overloaded without any issues.

```
class DisplayOverloading3 {
  public void disp(char c, int num)    {
     System.out.println("I’m the first definition of method disp");    }
  public void disp(int num, char c)    {
     System.out.println("I’m the second definition of method disp" );    } }
class Sample3 {
  public static void main(String args[])    {
     DisplayOverloading3 obj = new DisplayOverloading3();
     obj.disp('x', 51 );
     obj.disp(52, 'y');    } }
```

Output:

```
I’m the first definition of method disp
I’m the second definition of method disp
```

Method Overloading and Type Promotion

When a data type of smaller size is promoted to the data type of bigger size than this is called type promotion, for example: byte data type can be promoted to short, a short data type can be promoted to int, long, double etc.

H.W.

Define a class called B-Date which has 3 integer instance variables and two method set and print, the set method has been overloaded:

Set()

Set(int, int ,int)

Use the above class to create 2 objects and print the details of the older one according to its birth date year.