Lecture – 3

Constructors

Methods

- There are two kinds of methods:
  - Methods that compute and return a value

    ```
    public typeReturned methodName(paramList)
    ```

  - Methods that perform an action – `void` methods

    ```
        public void methodName(paramList)
    ```

Parameters of Methods

- When a method is invoked, the appropriate values must be passed to the method in the form of *arguments*
  - Arguments are also called *actual parameters*
- The number and order of the arguments must exactly match that of the parameter list
- The type of each argument must be compatible with the type of the corresponding parameter
- Parameters can of two types
  - Primitive Types: `public void sum(int a, int b)`
  - Reference Types: `public void showData(String name, Object obj)`

Overloading

- *Overloading* is when two or more methods *in the same class* have the same method name
- To be valid, any two definitions of the method name must have different *signatures*
  - A signature consists of the name of a method together with its parameter list
  - Differing signatures must have different numbers and/or types of parameters
  - ***Removing ambiguity by giving different parameters***

Overloading and Automatic Type Conversion

- If Java cannot find a method signature that exactly matches a method invocation, it will try to use automatic type conversion
- In some cases of overloading, because of automatic type conversion, a single method invocation can be resolved in multiple ways
  - Ambiguous method invocations will produce an error in Java

Example

```
class Adder{
    int add(int a,int b){
        return a+b;
    }
    double add(double a,double b){
        return a+b;
    }
}
class ExampleOverloading{
    public static void main(String[] args){
        Adder add = new Adder();
        System.out.println(add.add(11,11));
        System.out.println(add.add(11.0,11));
    }
}
```

Initializing Objects

- Initializing an object mean assigning values to its data members
- Example
  - To create a `Point` object and initialize it:

```
Point p = new Point();
p.x = 3;
p.y = 8;       // tedious
```

- We'd rather pass the fields' initial values as parameters:
  - `Point p = new Point(3, 8);  // better!`
  - We can do this with most types of objects in Java

Constructors

- A *constructor* is a special kind of method that is designed to initialize the instance variables(or state)for an object:

```
public ClassName(anyParameters){
        Code
}
```

  - A constructor must have the same name as the class
  - A constructor has no type returned, not even **`void`**
  - Constructors are typically overloaded
- Two types of constructors in Java:
  - no-arg constructor (Default Constructor)
  - Parameterized constructor

Constructors

- A constructor is called when an object of the class is created using **`new`**

```
ClassName objectName = new ClassName(anyArgs);
```

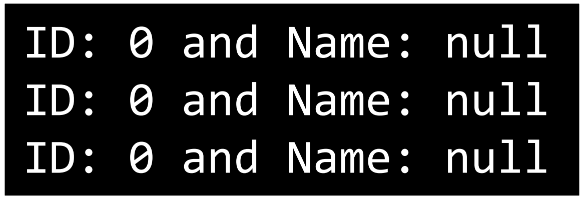
  - This is the **only** valid way to invoke a constructor
  - A constructor cannot be invoked like an ordinary method
- If a constructor is invoked again (using **`new`**), the first object is discarded and an entirely new object is created

```
    Student std1 = new Student(12,"Ali")
       std1 = new Student(13,"Hamza");
```

No-Argument Constructor

```
public class ConsDemo{
    public static void main(String args[]){
        Student std1 = new Student();
        Student std2 = new Student();
        Student std3 = new Student();
        std1.show();
        std2.show();
        std3.show();
    }
}
class Student{
    int id;
    String name;
    public void show(){
        System.out.println("ID: "+id+" and Name: "+name);
    }

}
```



```
ID: 0 and Name: null
ID: 0 and Name: null
ID: 0 and Name: null
```

Parameterized Constructor

```
public class ConsDemo{
    public static void main(String args[]){
        Student std1 = new Student(12,"Ali");
        Student std2 = new Student(13,"Hashir");
        Student std3 = new Student(14,"Ahmad");
        std1.display();
        std2.display();
        std3.display();
    } }
class Student{
    int id;
    String name;
    Student(int stdId, String stdName){
        id = stdId;
        name = stdName;
    }
    public void display(){
        System.out.println("ID: "+id+" and Name: "+name);
    }}
```

```
ID: 12 and Name: Ali
ID: 13 and Name: Hashir
ID: 14 and Name: Ahmad
```

Overloaded Constructor

```
public class ConsDemo{
    public static void main(String args[]){
        Student1 std1 = new Student1(12,"Ali");
        Student1 std2 = new Student1(13,"Hashir",23);
        std1.display();
        std2.display();
    }
}
class Student1{
    int id;
    String name;
    int age;
    Student1(int stdId, String stdName){
        id = stdId;
        name = stdName;
    }
    Student1(int stdId, String stdName, int stdAge){
        id = stdId;
        name = stdName;
        age = stdAge;
    }
    public void display(){
        System.out.println("ID: "+id+" Name: "+name+" Age: "+age);
    }
}
```

```
ID: 12 Name: Ali Age: 0
ID: 13 Name: Hashir Age: 23
```

Points About Constructor

- A default constructor has no parameters
- Java defines a constructor (default) if you do not define your own constructor
- Java will not provide default constructor, if you include even one constructor in your class
- If you include any constructors in your class, be sure to provide your own no-argument constructor
- You can invoke another method within the definition of a constructor

```
public Date(int monthInt, int day, int year){
    setDate(monthInt, day, year);
}
```

Points About Constructor

- In constructor, instance variables must have valid values.

```
public Student(String stdName, int stdAge, double stdMarks)
{
          name = stdName;
          if ((stdAge< 0) || (stdMarks< 0)){
                  System.out.println("Error: Negative age or marks.");
                  System.exit(0);
          }
          else{
                  age = stdAge;
                  marks= stdMarks;
          }
}
```

Common constructor bugs

- Re-declaring fields as local variables ("shadowing"):

```
public Point(int initialX, int initialY) {
        int x = initialX;
        int y = initialY;
    }
```

  - This declares local variables with the same name as the fields, rather than storing values into the fields. The fields remain 0.

- Accidentally giving the constructor a return type:

```
public void Point(int initialX, int initialY) {
        x = initialX;
        y = initialY;
    }
```

  - This is actually not a constructor, but a method named `Point`

- Missing Assignment to data:

```
public Point(int initialX, int initialY) {
        initialX = x;
        initialY = y;
}
```

  - This declares local variables with the same name as the fields, rather than storing values into the fields. The fields remain 0.

Exercise

- Create Time class with four overloaded constructors.
- Create three objects in the runner using any three of the constructors