OBJECT ORIENTED PROGRAMMING

User Defined Class II

by
Dr. Nor Saradatul Akmar Zulkifli
Faculty of Computer Systems & Software Engineering
saradatulakmar@ump.edu.my

EXAMPLE : Class Bicycle


```
class Bicycle {
    // Data member
    private String ownerName;

    //Constructor: Initializes the data member
    public Bicycle(){
        ownerName = "Unknown";
    }
    //Returns the name of this bicycle's owner
public String getOwnerName(){
    return ownerName;
}
//Assigns the name of this bicyle's owner
public void setOwnerName(String name){
    ownerName = name;
    }
}
```

Class Name

Data fields

Constructor and Member methods

EXAMPLE : MAIN CLASS


```java
class BicycleRegistration {

    public static void main(String[] args) {
        Bicycle bike1, bike2; //Create instances of class Bicycle
        String owner1, owner2;

        bike1 = new Bicycle(); //Create and assign values to bike1
        bike1.setOwnerName ("Sara Zulkifli");

        bike2 = new Bicycle(); //Create and assign values to bike2
        bike2.setOwnerName ("Ben Jones");

        owner1 = bike1.getOwnerName ( );  //Output the information
        owner2 = bike2.getOwnerName ( );
        boolean owner;

        System.out.println(owner1 + " owns a bicycle.");
        System.out.println(owner2 + " also owns a bicycle.");
    }
```

In previous lecture,

You have learned about:

- Class and object
- Anatomy of Class Definition
- Object, Class and method
- Invoking Method

Content Overview

- Constructor
- Accessor and Mutator (get and set)
- Matching Arguments and Parameters
- Passing and Returning Object to/from a method

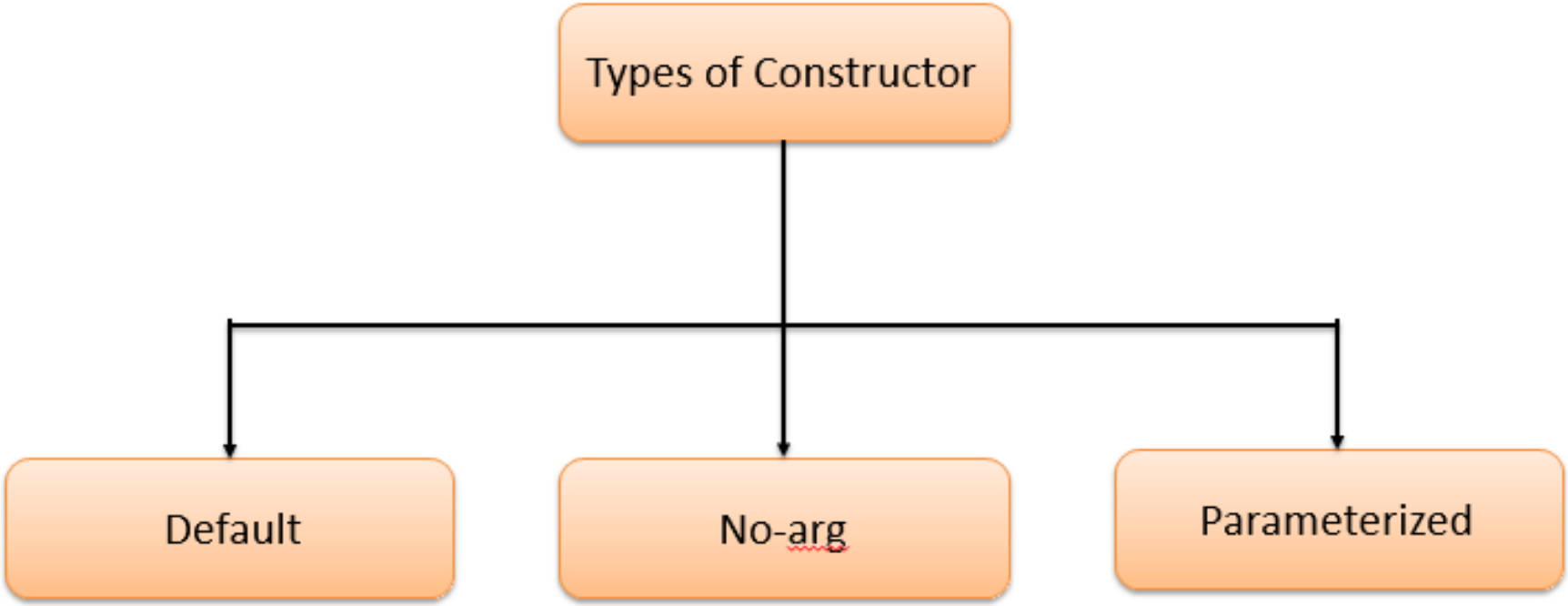
Learning Objectives

Student should be able to:

- To understand the concept of constructor

CONSTRUCTOR

- In Java, there are three types of constructors:



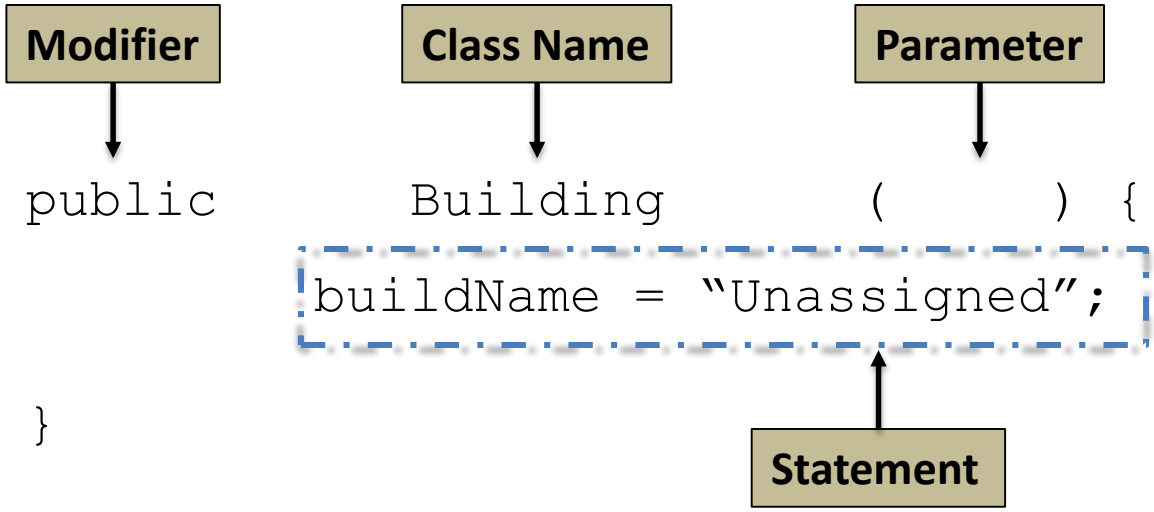
CONSTRUCTOR

- It is a special method used to create a new instance of the class

Syntax

```
public  <class name> (<parameters>) {
          <statements>
}
```

Example



CONSTRUCTOR

A constructor is a special method that have the following features:

- Similar name as the class
- Do not have a return type (not even void)
- It's initialize the default value for all object data members
- Invoked using the *new* operator when an object is created
- Each constructor must have a different number of parameters or parameters of different types.
- **Best Practice : Initialize all the object data member's value using constructor**

1. CONSTRUCTOR : DEFAULT

- A class may be defined without constructors
- Known as Default constructor – When no argument constructor with an empty body is implicitly declared in the class
- It's provided automatically only if no constructor are explicitly defined in the class.

Data Type	Default Value
byte, short, int, long	0
float, double	0.0
char	Space
boolean	False
Any object reference (e.g String)	Null

- The following default value will be automatically assigned if the constructor does not assign ant value to the instance variables

DEFAULT CONSTRUCTOR : EXAMPLE

- No constructor methods were defined in the Vehicles class
- No-argument constructor with an empty body is implicitly declared in the class

```java
public class Vehicles {
    String ownerName;
    int plateNo;
    int hour;

    public double calculateFine (){
        double fine;
        if (hour >3)
            fine = 10.0;
        else
            fine = 0.0;
        return fine;
    }

    public void displayInfo (){
        System.out.println ("Your Name: " +ownerName);
        System.out.println ("Your Car Registration No.: " +plateNo);
        System.out.println ("Hour of Parking: " +hour);
        System.out.println ("Fine: " +calculateFine());
    }
}
```

OUTPUT

Output - Vehicle (run)

```
run:
Your Name: null
Your Car Registration No.: 0
Hour of Parking: 0
Fine: 0.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

DEFAULT CONSTRUCTOR : EXAMPLE

- Object Instantiation using default constructor

```
public class MainVehicle {

    public static void main(String[] args) {
        Vehicles car = new Vehicles();
        car.displayInfo();
    }

}
```

Vehicles

ownerName : String
plateNo : int
hour : int

Class

car

ownerName : null
plateNo : 0
hour : 0

Object

2. NO-ARG CONSTRUCTOR

- No-arg constructor is the same as default constructor HOWEVER, body statement can have any code

- Compare to Default constructor which the body is empty.

No Argument

```
public Vehicles(){
    System.out.println ("This is no-arg constructor");
}
```

Body Statement

3. PARAMETERIZED CONSTRUCTOR : EXAMPLE

```java
public class Vehicles {
    String ownerName;
    int plateNo;
    int hour;

    public Vehicles (String ownName, int ownPlate){
        ownerName = ownName;
        plateNo = ownPlate;
    }

    public Vehicles (String ownName, int ownPlate, int hour){
        this.ownerName = ownName;
        this.plateNo = ownPlate;
        this.hour = hour;
    }
    public double calculateFine (){
        double fine;
        if (hour >3)
            fine = 10.0;
        else
            fine = 0.0;
        return fine;
    }
```

Parameters

OUTPUT

```
Output - Vehicle (run)
run:
Your Name: Proton
Your Car Registration No.: 12345
Hour of Parking: 0
Fine: 0.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

3. PARAMETERIZED CONSTRUCTOR : EXAMPLE

```
public class MainVehicle {

    public static void main(String[] args) {
        Vehicles car = new Vehicles("Proton", 12345);
        car.displayInfo();
    }
}
```

Arguments

Class

Vehicles
ownerName : String plateNo : int hour : int

- Object Instantiation using default constructor

Object

Vehicles : car
ownerName : “Proton” plateNo : 12345 hour : 0

3. PARAMETERIZED CONSTRUCTOR : ARGUMENTS VS. PARAMETER

- An argument is a value we pass to a method
- A parameter is a placeholder in the called method to hold the value of the passed argument

Parameters

```java
public class Vehicles {
    String ownerName;
    int plateNo;
    int hour;

    public Vehicles (String ownName, int ownPlate){
        ownerName = ownName;
        plateNo = ownPlate;
    }

    public Vehicles (String ownName, int ownPlate, int hour){
        this.ownerName = ownName;
        this.plateNo = ownPlate;
        this.hour = hour;
    }
```

Arguments

```java
public class MainVehicle {

    public static void main(String[] args) {
        Vehicles car = new Vehicles("Proton", 12345);
        car.displayInfo();
    }
}
```

3. PARAMETERIZED CONSTRUCTOR : ARGUMENTS VS. PARAMETER

- The number of arguments and the parameters must be the same.
- Arguments and parameters are paired from left to right

HINT :

If you have two or more constructors (**overloading constructors**), let say a default constructor and a parameterized constructor.

- ✓ When you do not pass any parameter while creating the object using `new` keyword, then default constructor is invoked
- ✓ When a parameter is pass during object instantiation, then parameterized constructor that MATCHES with the passed parameters list gets invoked.

CONSTRUCTOR : this

- This codes using the similar name for **instance variables** and **parameters**,
- Causing in compiler treat the variables inside the method as parameters


```
public Vehicles (String ownName, int ownPlate){ //constructor
    ownerName = ownName;
    plateNo = ownPlate;
}

public Vehicles (String ownerName, int plateNo, int hour){ //Constructor
    ownerName = ownerName;
    plateNo = plateNo;
    hour = hour;
}
```

Solution:

1. Use the different name for the instance variable and parameter
2. Use **this** to refer to an instance variable

CONSTRUCTOR : this

- **this** is an implicit parameter sent to methods and is an object reference to the object for which the method was called

1. **Use the different name for the instance variable and parameter**

```
public Vehicles (String ownName, int ownPlate){
    ownerName = ownName;
    plateNo = ownPlate;
}
```

2. **Use this to refer to an instance variable**

```
public Vehicles (String ownName, int ownPlate, int hour){
    this.ownerName = ownName;
    this.plateNo = ownPlate;
    this.hour = hour;
}
```

CONSTRUCTOR : this

- ✓ If there a several constructors in a system, one constructor is calling another constructor using **this** keyword
- ✓ **this ( )** should always be the 1st statement in constructor, or else an error message will appear.

```java
public class Vehicles {
    String ownerName;
    int plateNo;
    int hour;
    //default constructor of the class
    public Vehicles(){
        //this will call the constructor with String param
        this ("Ammar");
    }

    public Vehicles(String ownName){
        //call the constructor with (String, int) param
        this(ownName,998022);
    }
    public Vehicles (String ownName, int ownPlate){
        //call the constructor with String, int, int) param
        this(ownName,ownPlate,3);
    }

    public Vehicles (String ownName, int ownPlate, int hour){
        this.ownerName = ownName;
        this.plateNo = ownPlate;
        this.hour = hour;
    }
```

Output - Vehicle (run)

```
run:
Your Name: Ammar
Your Car Registration No.: 998022
Hour of Parking: 3
Fine: 0.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

CONSTRUCTOR CHAINING

Beginnersbook.com

```
public class MyClass{
    ....
    MyClass( ) {
        this("BeginnersBook.com");
    }
    MyClass(String s) {
        this(s, 6);
    }
    MyClass(String s, int age) {
        this.name =s;
        this.age = age;
    }
    public static void main(String args[]) {
        MyClass obj = new MyClass();
        ....
    }
}
```

- When a constructor is invoked from the other constructor of same class
- **WHY?**
  - A programmer can pass parameters through a bunch of different constructors, but only have the initialization done in a single place.
  - RULE : Constructor with most fewer arguments should call those with more

Source: https://beginnersbook.com/2013/12/java-constructor-chaining-with-example/

ACCESSING CLASS MEMBERS

Universiti Malaysia PAHANG

Syntax to ACCESS data

```
<objectName> . <instanceVariable>;
```

Syntax to ACCESS or INVOKE method

```
<objectName> . <methodName (argument(s))>;
```

Use the dot operator (.) to access its data or invoke its methods after an object is created

ACCESSING CLASS MEMBERS : EXAMPLE

```java
public class Vehicles {
    String ownerName;
    int plateNo;
    int hour;

    public Vehicles (String ownName, int ownPlate){ //constructor
        ownerName = ownName;
        plateNo = ownPlate;
    }

    public Vehicles (String ownerName, int plateNo, int hour){ //Constructor
        ownerName = ownerName;
        plateNo = plateNo;
        hour = hour;
    }

    public double calculateFine (){
        double fine;
        if (hour >3)
            fine = 10.0;
        else
            fine = 0.0;
        return fine;
    }
```

```java
public class MainVehicle {

    public static void main(String[] args) {
        Vehicles car = new Vehicles("Ammar", 89201);
        car.hour = 6;
        car.displayInfo();
    }
}
```

Accessing data

ACCESSING CLASS MEMBERS : EXAMPLE

Universiti Malaysia PAHANG

```
public double calculateFine (){
    double fine;
    if (hour >3)
        fine = 10.0;
    else
        fine = 0.0;
    return fine;
}

public void displayInfo (){
    System.out.println ("Your Name: " +ownerName);
    System.out.println ("Your Car Registration No.: " +plateNo);
    System.out.println ("Hour of Parking: " +hour);
    System.out.println ("Fine: " +calculateFine());
}
```

Invoking method

```
public class MainVehicle {

    public static void main(String[] args) {
        Vehicles car = new Vehicles("Ammar", 89201);
        car.hour = 6;
        car.displayInfo();
    }
}
```

ACCESSING CLASS MEMBERS : EXAMPLE

Output - Vehicle (run)

```
run:
Your Name: Ammar
Your Car Registration No.: 89201
Hour of Parking: 6
Fine: 10.0
BUILD SUCCESSFUL (total time: 0 seconds)
```

ACCESSOR AND MUTATOR

- **WHY?**
  - ✓ To enforce data encapsulation
  - ✓ Its allow programmer to change how the data is handled behind the scene
  - ✓ Enable programmer to impose validation on the values that the fields are being set to.
- **HOW?**
  - ✓ By returning and setting the values of an object's state (from private field)

ACCESSOR vs. MUTATOR

Universiti Malaysia PAHANG

ACCESSOR METHOD

- A method used to return the information of a private field
- Keyword : `get`
- It will always return the similar data type as their corresponding private field and simply return the information of that private field.

MUTATOR METHOD

- A method used to set a property/value of a private field
- Keyword : `set`
- Do not have a return type
- Accept parameter that has similar data type to the corresponding private field

ACCESSOR : EXAMPLE

```
public class Person {

  //Private fields
  private String firstName;
  private String middleNames;
  private String lastName;
  private String address;
  private String username;

  //Constructor method
  public Person(String firstName, String middleNames,
String lastName, String address)
  {
    this.firstName = firstName;
    this.middleNames = middleNames;
    this.lastName = lastName;
    this.address = address;
    this.username = "";
  }
}
```

```
//Accessor for firstName
public String getFirstName()
{
  return firstName;
}

//Accessor for middleNames
public String getMiddlesNames()
{
  return middleNames;
}

//Accessor for lastName
public String getLastName()
{
  return lastName;
}
```

Source: https://www.thoughtco.com/accessors-and-mutators-2034335

ACCESSOR : EXAMPLE

Main class

```
public class PersonExample {

  public static void main(String[] args) {

    Person dave = new Person("Dave", "Bob Bill",
"Davidson", "12 Pall Mall");
    System.out.println(dave.getFirstName() + " " +
dave.getMiddlesNames() + " " + dave.getLastName());
  }
}
```

Source: https://www.thoughtco.com/accessors-and-mutators-2034335

MUTATOR: EXAMPLE

```
public class Person {

  //Private fields
  private String firstName;
  private String middleNames;
  private String lastName;
  private String address;
  private String username;

  //Constructor method
  public Person(String firstName, String middleNames,
String lastName, String address)
  {
    this.firstName = firstName;
    this.middleNames = middleNames;
    this.lastName = lastName;
    this.address = address;
    this.username = "";
  }
}
```

```
//Mutator for address
public void setAddress(String address)
{
  this.address = address;
}

//Mutator for username
public void setUsername(String username)
{
  this.username = username;
}
```

Source: https://www.thoughtco.com/accessors-and-mutators-2034335

MUTATOR: EXAMPLE

Main class

```
public class PersonExample {

  public static void main(String[] args) {

    Person dave = new Person("Dave", "Bob Bill",
"Davidson", "12 Pall Mall");
    dave.setAddress("256 Bow Street");
    dave.setUsername("DDavidson");

  }
}
```

It's enable programmer to modify address and username inside the `Person` object

Source: https://www.thoughtco.com/accessors-and-mutators-2034335

ARGUMENTS

- A value we pass to a method
- The value is assigned to the corresponding parameters
- Called as an actual parameter

```
class Sample {

    public static void
            main(String[] arg) {

      Account acct = new Account();
      . . .

      acct.add(400);
      . . .
    }

    . . .
}
```

argument

PARAMETERS

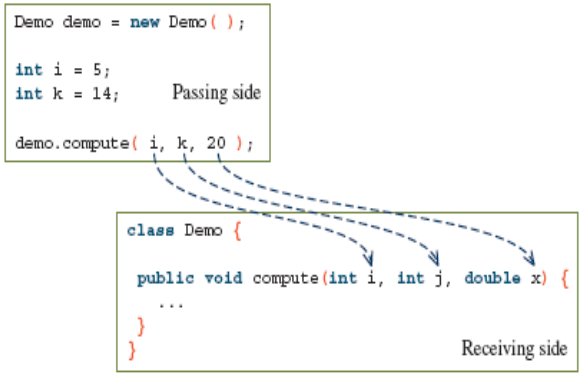
- A placeholder in the called method to hold the value of the passed arguments
- Called as formal parameters

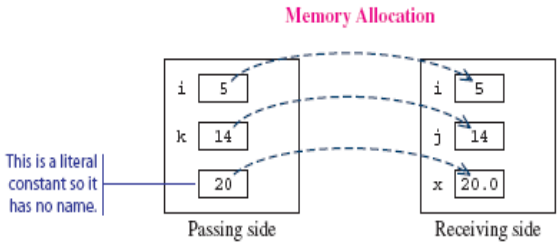
```
class Account {

    . . .

    public void add(double amt) {

        balance = balance + amt;
    }

    . . .
}
```

parameter

MATCHING ARGUMENTS AND PARAMETERS

- The number of arguments and the parameters must be the same.
- The matched pair must be assignment-compatible (e.g: you cannot pass a double argument to an int parameter)
- Arguments and parameters are paired from left to right





A parameter receives the value of a corresponding arguments – because a parameter is like a placeholder that will not hold a value until an arguments is passed to it.

Source : Thomas Wu
Introduction to Object Oriented Programming

PASSING OBJECT TO METHOD

There are two ways to pass argument(s) to methods:

I. Passing by value for primitive type variable – the value is passed to the parameter.

II. Passing by value for **reference type variable** – the value is the reference to the object.

Allows an object to be referred multiple times

Passing by value for primitive type variable

MOST COMMON TO BE USED

PASSING OBJECT TO METHOD : EXAMPLE

There

Author Information

Dr. Nor Saradatul Akmar Binti Zulkifli

Senior Lecturer
Faculty of Computer Systems & Software Engineering
Universiti Malaysia Pahang