Object Oriented Programming

ICT2122

Classes and Objects

P.H.P. Nuwan Laksiri
Department of ICT
Faculty of Technology
University of Ruhuna

Lesson 02 – Part 02

Recap

- Object Oriented Programming - Concepts
- Understanding Objects
- Understanding Classes
- Understanding Fields
- Understanding Methods
- JAVA - Access Modifiers
- Creating Objects
- Initializing Objects
  - By reference variable
  - By method
  - By constructor
- Understanding Constructors
  - Default
  - Parameterized

Outline

- *JAVA – this keyword*
  - methods
  - constructors
- *JAVA - Constructor Chaining*
- *JAVA - Anonymous objects*
- *JAVA – Garbage Collection*
- *Static in Java*
  - *Static Fields*
  - *Static Methods*
  - *Static Initializers*
- Preventing instantiating a class

Have you tried ???

```
class Account
{
        int a,b;
        public void setData(int a, int b)
        {
                a=a;
                b=b;
        }
        public void showData(){
                System.out.println(“Value of A=” +a);
                System.out.println(“Value of B=” +b);
        }
        public static void main(String[] args)
        {
                Account myAccount= new Account();
                 myAccount.setData(2,3);
                 myAccount.showData();
        }
}
```

Have you tried ???

- Why?
  - Both local and instance variables are same.
- Solution???
  - The "this" reference
  - Every object has a reference to itself represented by the "this" keyword
- Change code segment to

```
public void setData(int a, int b){
    this.a=a;
    this.b=b;
 }
```

- In the compilation time "this" will replace with "myAccount".
  - Then left-hand side variable becomes the instance variable, and the right-hand side variable becomes the local variable.

Uses of “this” keyword

- Can use ***this*** in the body of a class constructor or method to refer to the current object
    - that is, the class instance for which the constructor or method has been called.

Uses for “this” keyword - methods

```
Instance variables
String firstName, lastName;

public void setNames(String last, String first)          →          Required ???
{
            this.lastName = last;
            this.firstName = first;
}

public void setNames(String lastName, String firstName)  →          Required ???
{
            this.lastName = lastName;
            this.firstName = firstName;
}

public String getFullName()                              →          Required ???
{
return this.firstName + " " + this.lastName;
}

System.out.println(this);                                →          What happens???
```

Uses for “this” keyword - Constructors

- Can call another constructor only in the very first statement of a constructor by using “this” keyword
- Each constructor can call only one other constructor, but you can chain constructors
- You can’t create loops in which constructors call one another

Hands-on

  - Let’s try

JAVA - Constructor Chaining

- Constructor chaining refers to the ability to call a constructor inside another constructor.
  - You can use a constructor chain either within the same class or even with another one.
- JAVA - Constructor Chaining in the Same Class
  - Using "this" keyword
  - Discussed above
- JAVA - Constructor Chaining to Another Class
  - Using "super" keyword
  - Will be discussed under Inheritance

JAVA – Initializer (Initializer block)

- Initializer block is a lonely block of code that's placed outside any method, constructor, or other block of code.
- Initializers are executed whenever an instance of a class is created, regardless of which constructor is used to create the instance.

```
Public class InitializerTest
{
    {
                System.out.print("Initializer block executed…!!!");
    }
}
```

JAVA – Initializer (Initializer block)

- If a class contains more than one initializer, the initializers are executed in the order in which they appear in the program.
- Initializers are executed before any class constructors.
- A special kind of initializer block called a static initializer lets you initialize static fields.
- Initializers are sometimes used with anonymous classes.
  - ???

Hands-on

- Let's try some

JAVA - Anonymous objects

- Anonymous simply means nameless.
- An object which has no reference is known as anonymous object.
- It can be used at the time of object creation only.
- If you have to use an object only once, anonymous object is a good approach.

  Ex:

  ```
  new Calculation();//anonymous object
  ```

JAVA - Anonymous objects

- Calling method through reference,

```
Calculation c=new Calculation();
c.fact(5);
```

- Calling method through anonymous object,

```
new Calculation().fact(5);
```

How can an object be unreferenced?

- By nulling the reference

```
Employee e=new Employee();
e=null;
```

- By assigning a reference to another

```
Employee e1=new Employee();
Employee e2=new Employee();
e1=e2;//now the first object referred by
  e1  is available for garbage   collection
```

- By annonymous object etc.

```
new Employee();
```

JAVA - Garbage Collection

- In java, garbage means unreferenced objects.
- Garbage Collection is process of reclaiming the runtime unused memory automatically.
  - In other words, it is a way to destroy the unused objects.
- The Java runtime environment deletes objects when it determines that they are no longer being used.
- In java it is performed automatically. So, java provides better memory management.

Advantages of JAVA Garbage Collector

- It makes java memory efficient because garbage collector removes the unreferenced objects from heap memory.

- It is automatically done by the garbage collector(a part of JVM) so we don't need to make extra efforts.

Homework

- What are the disadvantages of JAVA- Garbage Collectoer

JAVA - Working with Statics

What does the term **static** mean in Java?

- It's used to describe a special type of field or method that isn't associated with a particular instance of a class.
  - static fields and methods are associated with the class itself

- You don't have to create an instance of the class to access a static field or methods
  - You can access a static field or method by specifying the class name, not a variable that references an object

Common Uses of Static fields and Methods in Java

- To provide constants or other values that aren't related to class instances
- To keep count of how many instances of a class have been created
- To keep track of a reference or serial number that's assigned to each new object instance
- To provide an alternative way to create instances of the class
- To provide utility functions that aren't associated with an object at all

Static Fields

- A static field is a field that's declared with the static keyword
  - Ex : private static int age;
  - <access modifier> static <datatype> <field>

- You can't use the static keyword within a class method
  - Code won't compile

- Fields can be static, but local variables can't

Static Fields

- You can provide an initial value for a static field
  - Ex : private static int age = 20;
- Static fields are created and initialized when the class is first loaded
  - when a static member of the class is referred to or
  - when an instance of the class is created (whichever comes first)
  - using a static initializer

Static Methods

- A static method is a method declared with the static keyword
  - Like static fields, static methods are associated with the class itself, not with any particular object created from the class
- The best-known static method is ***main***
  - Called by the Java runtime to start an application
  - The Java applications are run in a static context by default

Static Methods

- You can’t access a non static method or field from a static method
  - The static method doesn’t have an instance of the class to use to reference instance methods or fields
  - But You can access static methods and fields from an instance method

Hands-on

- Static Fields

- Static Methods

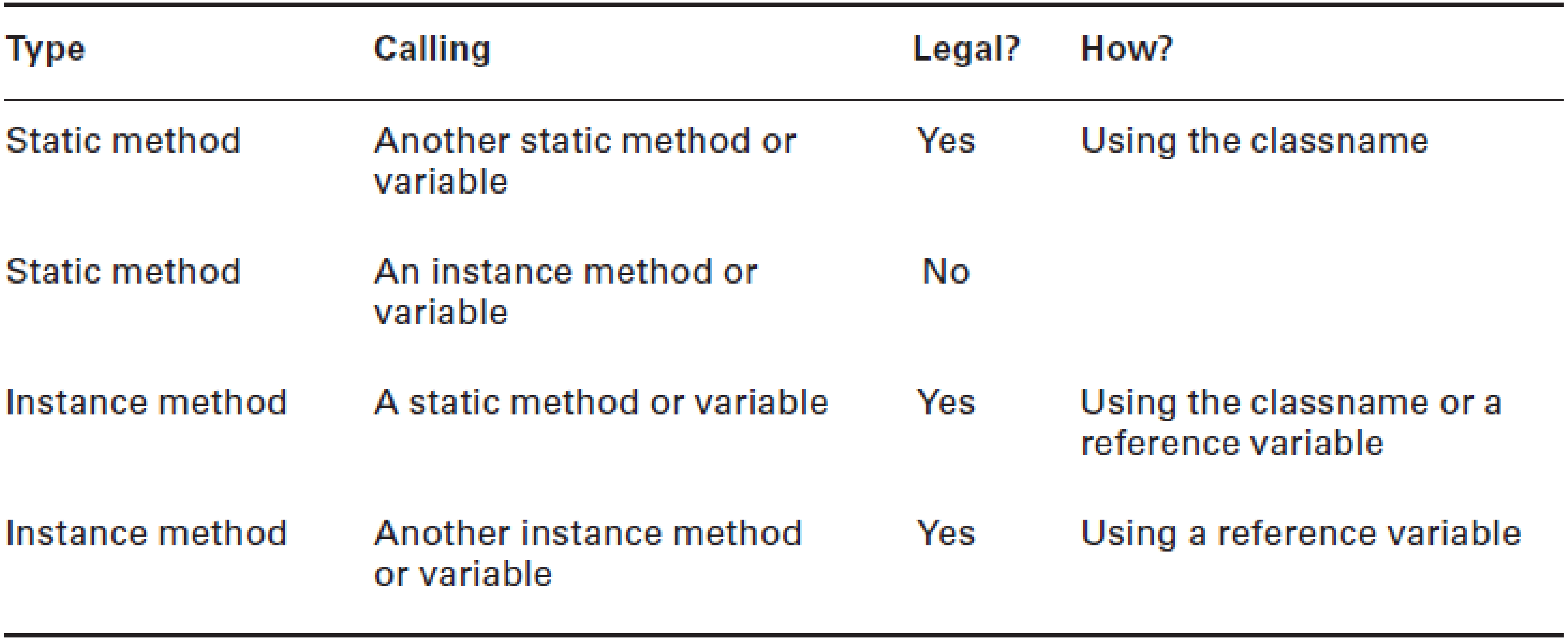
Counting Instances with Static

- One common use for static variables is to keep track of how many instances of a class have been created
  - instance count in the hands-on session application is reset to zero each time the application is run
  - It doesn’t keep track of how many instances of the class have ever been created
    - Keep only of how many have been created during a particular execution of the program

Hands on - Counting Instances with Static

What can call what?

Static vs. instance calls



Type	Calling	Legal?	How?
Static method	Another static method or variable	Yes	Using the classname
Static method	An instance method or variable	No	
Instance method	A static method or variable	Yes	Using the classname or a reference variable
Instance method	Another instance method or variable	Yes	Using a reference variable

Static Initializers

- Java provides a feature called a static initializer that's designed specifically to let you initialize static fields.

```
static
{
        statements
 }
```

- Initializer block begins with the word static.
- You can have static initializers in the class body
  - outside any other block, such as the body of a method or constructor

Static Initializers

- The first time you access a static member such as a static field or a static method, any static initializers in the class are executed
    - static initializers are also executed the first time you create an instance
        - the static initializers are executed before the constructor is executed
- If a class has more than one static initializer
    - the initializers are executed in the order in which they appear in the program

Hands on- Static Initializers

Preventing instantiating a class

- To create a class instance, you have to have at least one public constructor.
  - If you don't provide a constructor in your class, Java automatically inserts a default constructor, which happens to be public
- All you have to do **to prevent a class instance from being created is provide a *single private constructor***

```
public class MyClass
{
    private MyClass() {} // prevents instances
    // static methods and fields go here
}
```

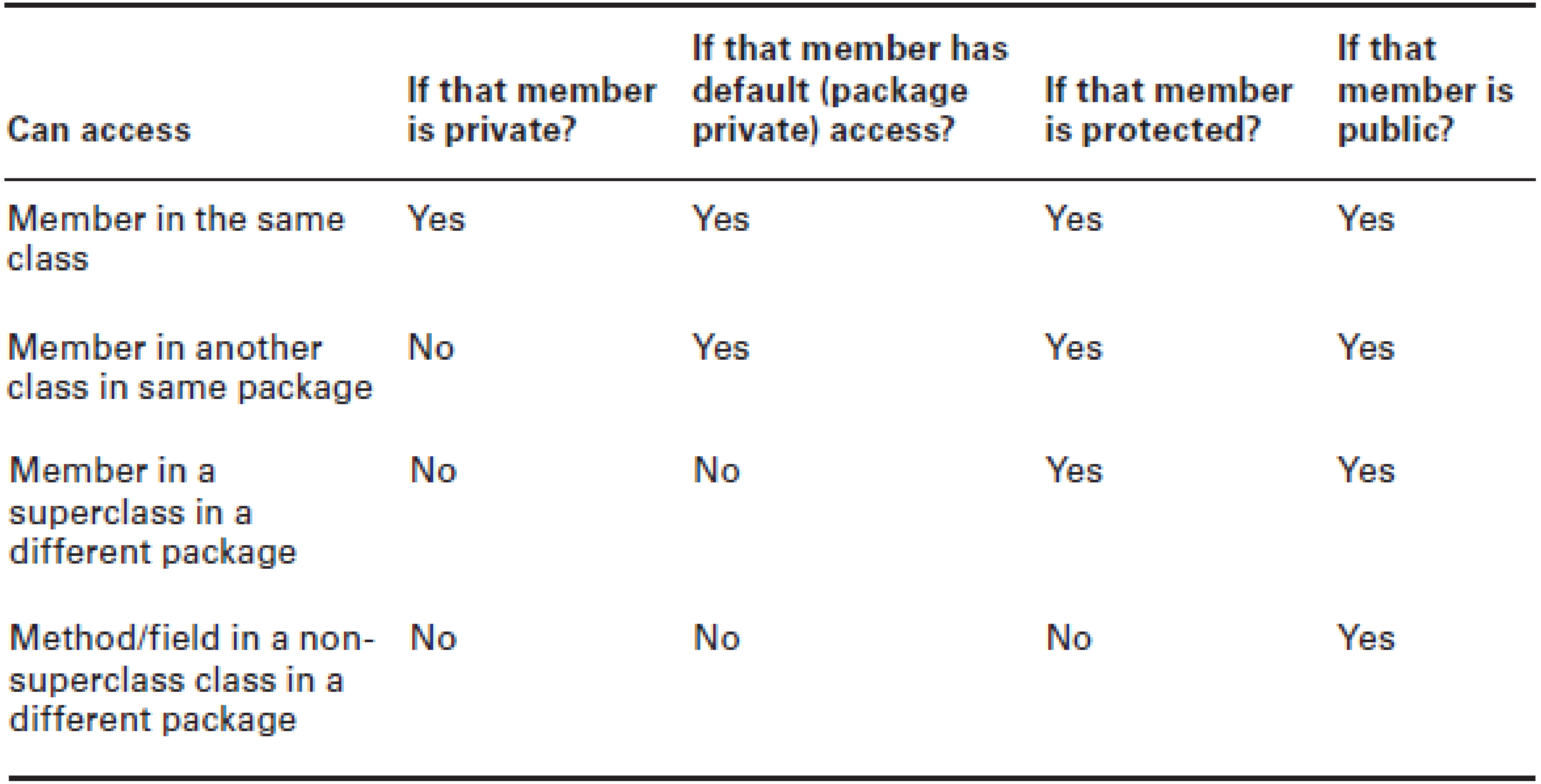
- Now, because the constructor is private, the class can't be instantiated

Hands on - Preventing instantiating a class

- Why we need to prevent instantiating a class
  - Ex : Math Class

- Singleton Design ???

Summary of Access Modifiers



Can access	If that member is private?	If that member has default (package private) access?	If that member is protected?	If that member is public?
Member in the same class	Yes	Yes	Yes	Yes
Member in another class in same package	No	Yes	Yes	Yes
Member in a superclass in a different package	No	No	Yes	Yes
Method/field in a non-superclass class in a different package	No	No	No	Yes

Homework

Order of Initialization – tryout your own coding

- If there is a superclass, initialize it first
- Static variable declarations and static initializers in the order they appear in the file.
- Instance variable declarations and instance initializers in the order they appear in the file.
- The constructor.

Quiz 01 – Next week

- Date : 20th February 2025
- Time : 08.10 a.m. to 08.30 a.m.
- Lessons
  - Lesson 01 – Introduction to OOP
  - Lesson 02 – Classes and Objects
    Part 01 and Part 02

Feedback

- #CleanNuwan

Summary

- *JAVA – this keyword*
  - methods
  - constructors
- *JAVA - Constructor Chaining*
- *JAVA - Anonymous objects*
- *JAVA – Garbage Collection*
- *Static in Java*
  - *Static Fields*
  - *Static Methods*
  - *Static Initializers*
- Preventing instantiating a class

References

- https://docs.oracle.com/javase/tutorial/java/javaOO/thiskey.html

- How To Program (Early Objects)
  - By H .Deitel and P. Deitel
- Headfirst Java
  - By Kathy Sierra and Bert Bates

Questions ???

Thank You