# Object Oriented Development with Java

(CT038-3-2 and Version VC1)


## Overview of OOP

with Java classes

# Topic & Structure of the lesson

- Objects
- Methods
- Accessors and Mutators
- Constructors
- Static members
- Visibility modifiers

# Learning outcomes

At the end of this lecture you should be able to:

- Understand the process of implementing Java classes
- Understand and implement the concept of methods

# Key terms you must be able to use

If you have mastered this topic, you should be able to use the following terms correctly in your assessments:

- Objects
- Classes
- Dynamic Binding
- Message Passing

# Defining a class

- In a Java program, everything is encapsulated in a class
- A class is a user defined data type
- Class defines the state and behavior of objects
- Once class type has been defined, we can create “variables” of that type
- These are known as instances of a class

# Defining a class

- The basic form of class definition is:

```
class classname [extends superclassname]
{
  [ variable declaration; ]
  [ methods declaration; ]
}
```

# Adding variables

- Instance variables are created whenever an object of the class is instantiated.
- Example:

```
class Rectangle
{
    int length;
    int width;
}
```

# Adding methods

- A class without methods that operate on data has no “life”.
- The objects created by such a class cannot respond to any messages.
- Therefore add methods that are necessary for manipulating data contained in the class.

- The general form of method declaration is:

```
type methodname (parameter-list)
{
    method-body;
}
```

- Example:

```
class Rectangle
{
        int length;
        int width;

        void getData (int x, int y)
        {
                length = x;
                width  = y;
        }
}
```

•Example:

Assume we want to compute the area of the rectangle.

```
class Rectangle
{
            int length, width;

            void getData (int x, int y)
            {
                  length = x;
                  width  = y;
            }
            int rectArea( )
            {
                  int area = length * width;
                  return (area);
            }
}
```

# Creating Objects

- Creating an object is also referred to as instantiating an object.
- Objects in Java are created using the new operator.
- The new operator creates an object of the specified class and returns a reference to the object.

# Example:

```
Rectangle rect1;              // declare
rect1 = new Rectangle( ); // instantiate
```

OR

```
Rectangle rect1 = new Rectangle ( );
```

# Accessing class members

The general form:

```
objectname. variable name
objectname. methodname (parameter-list);
```

# Accessing instance variables: Example



- The instance variables of the Rectangle class may be accessed and assigned values as follows:

```
rect1.length = 15;
rect1.width  = 10;
rect2.length = 20;
rect2.width  = 12;
```

# Assigning values to instance variables

- Another convenient way of assigning values to the instance variables is to use a method that is declared inside the class.

Example:

```
Rectangle rect1 = new Rectangle( ); //creating an object
rect1.getData(15,10); // calling the method using the object
```

# Using instance variables

- To compute the area of the rectangle represented by rect1 can be done in two ways.

```
int area1 = rect1.length * rect1.width;
```

OR

```
int area1 = rect1.rectArea();
```

# Application of classes and objects

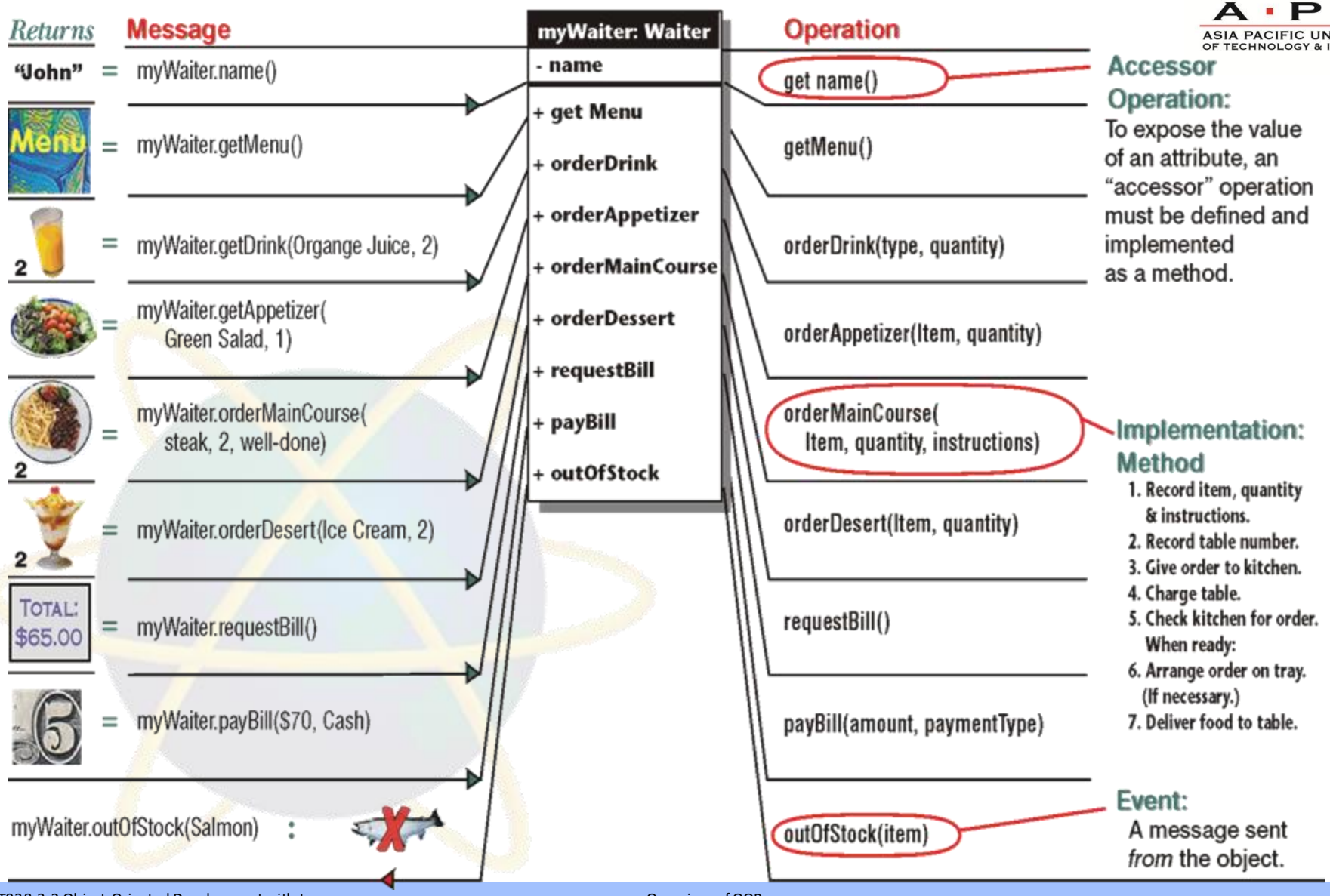
```
class Rectangle
{
        int length, width;

        void getData (int x, int y)
        {
                length = x;
                width  = y;
        }
        int rectArea( )
        {
                int area = length * width;
                return (area);
        }
}
```

(contd. on next page)

```
class RectArea {

   public static void main (String args[ ]) {

   int area1,area2;
   Rectangle rect1 = new Rectangle();
   Rectangle rect2 = new Rectangle();

   rect1.length = 15;
   rect1.width = 10;

   area1 = rect1.length * rect1.width;

   rect2.getData(20,12);
   area2 = rect2.rectArea();

   System.out.println("Area1 = " + area1);
   System.out.println("Area2 = " + area2);
   }
}
```

# Methods & Messages

## How Objects Interact




**Accessor Operation:**
To expose the value of an attribute, an "accessor" operation must be defined and implemented as a method.

**Implementation: Method**

1. Record item, quantity & instructions.
2. Record table number.
3. Give order to kitchen.
4. Charge table.
5. Check kitchen for order. When ready:
6. Arrange order on tray. (If necessary.)
7. Deliver food to table.

**Event:**
A message sent *from* the object.

# Accessor & Mutator Methods

- Enforce data encapsulation
- Hide data
- Able to change how data is handled
- Allows values set in attributes to be validated
- Also known as setter and getter methods

# Accessor & Mutator Methods

- Accessor method:
  - Returns/obtains value of a private attribute/variable
  - Method name starts with the prefix “get”
  - Example:

```
public String getColor() {
        return color;
}
```

  - To access the value:

```
Person Jerry = new Person();
System.out.println(Jerry.getColor());
```

# Accessor & Mutator Methods

- Mutator method:
  - Sets/assigns the value of a private attribute/variable
  - Method name starts with the prefix "set"
  - Example:

```
void setColor(String c) {
        color = c;
}
```

  - To set the value:

```
Person Jerry = new Person();
Jerry.setColor("Blue");
```

# Constructors

- Java supports a special type of method called a constructor.
- It enables an object to initialize itself when it is created.
- A constructor has the same name as the class itself.
- Do not have a return type because they return instances of the class.

# Consider the Rectangle class again. Replace the **getData** method by a constructor.

```
class Rectangle
{
      int length, width;

      Rectangle (int x, int y) // constructor method
      {
            length = x;
            width  = y;
      }
      int rectArea( )
      {
            return (length * width);
      }
}
```

# Static Members

- Used to define a member that is common to all the objects and accessed without using a particular object
  - member belongs to the class as a whole rather than the objects that were created from the class
- Such members can be defined as follows:

```
static int count;
static int max (int x, int y);
```

# Static Members contd.

- static variables are often referred to as class variables
- static methods are often referred to as class methods
- static variables and static methods can be called without using objects - are called using class names

```
class MathOperation {
    static float mul (float a, float b) {
        return a*b;    }
    static float divide (float a, float b) {
    return a/b;   }
};
class MathApplication {
    public static void main (String [] args) {
        float a = MathOperation.mul (4.0, 5.0);
        float b = MathOperation.divide(a, 2.0);
        System.out.println ("b=" +b);
    }
}
```

# Static Members contd.

Note:

1. Static methods are called using class names
2. No objects are created for use.

Restrictions:

1. They can only call other static methods.
2. They can only access static data.

# Visibility modifiers

- also known as access modifiers
- are applied to the instance variables and methods to restrict the access to these variables and methods from outside the class
- 3 types:
  - public
  - private
  - protected

# public Access

- any variable or method that is visible to all classes outside this class
- declared as:

```
public int number;
public void sum ( )
{
  ..................
}
```

- has the widest possible visibility and accessible everywhere
- classes usually define methods to be public so that the client program (eg. main

# friendly Access

- when NO access modifier is specified, the member defaults to a limited version of public accessibility known as friendly level of access
- makes fields visible in all classes but only in the same package but not in other packages
  - a package
    - is a group of related classes stored separately

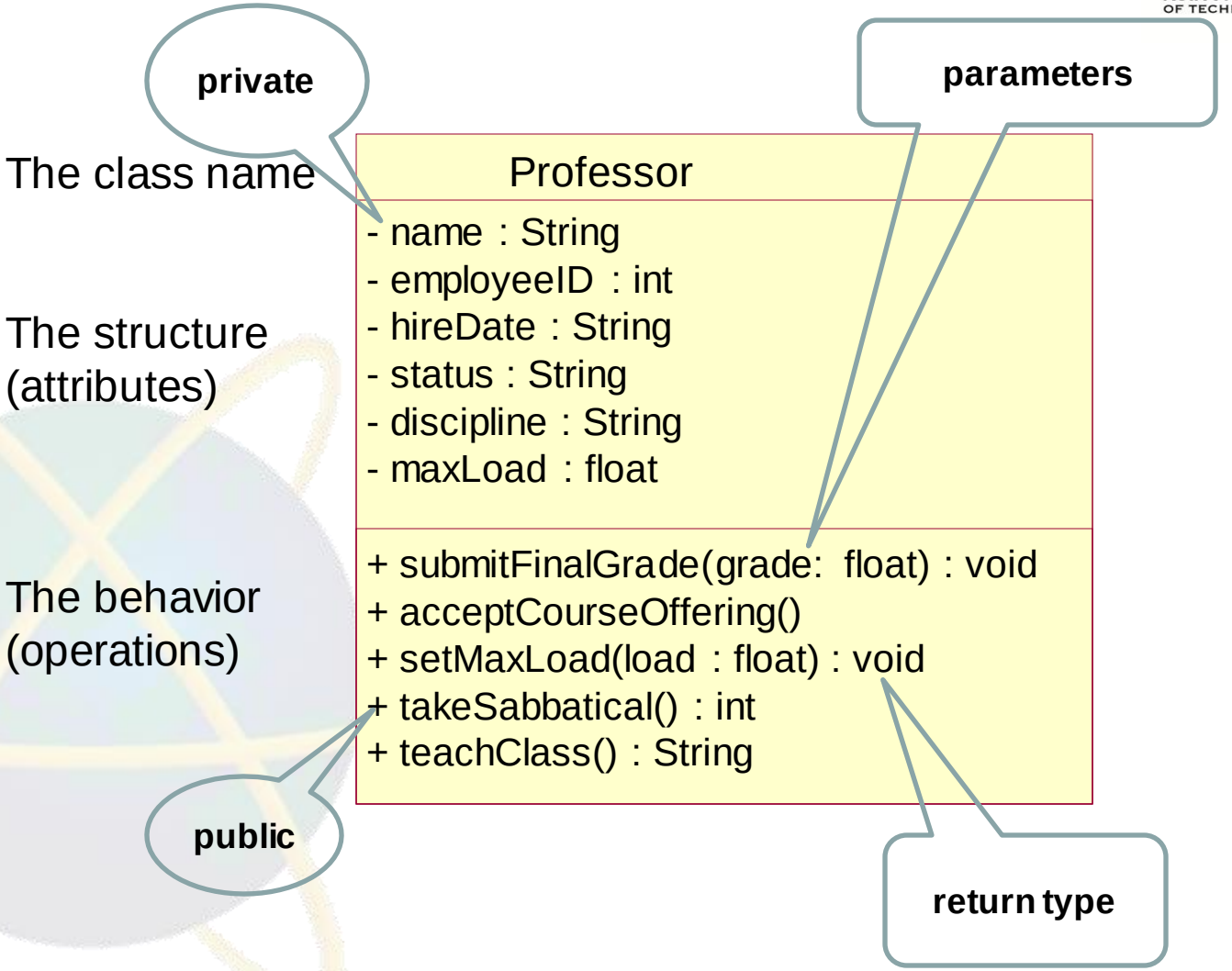
# protected Access

- visibility level lies between the public access and friendly access
- makes the fields visible not only to all classes and subclasses in the same package but also to subclasses in other packages
- non-subclasses in other packages cannot access the protected members

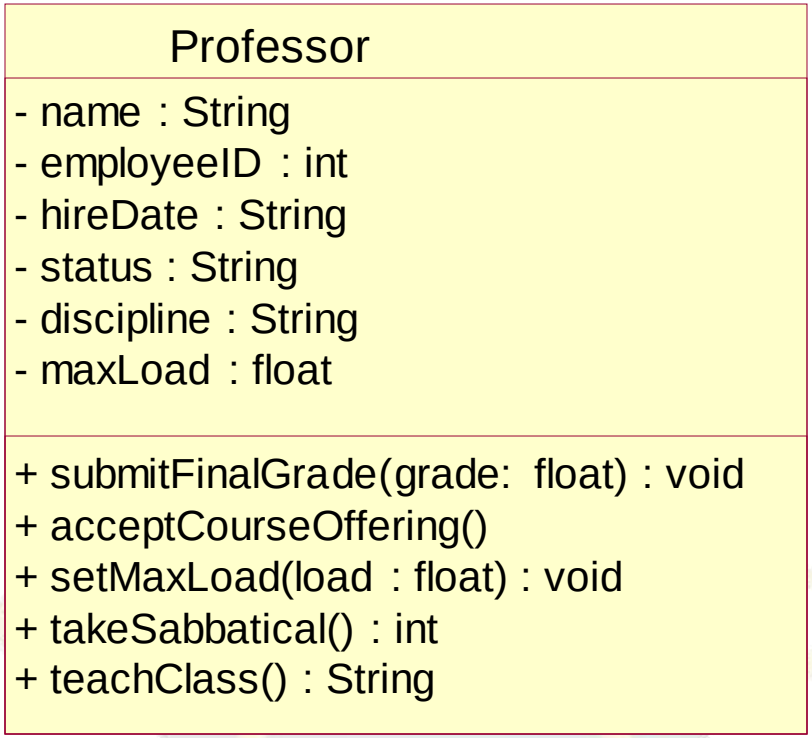
# private Access

- private access fields
  - have highest degree of protection
  - are accessible only within their own class
  - cannot be inherited by subclass
  - is not accessible in subclass
- private access methods
  - behaves like a method declared as final
  - prevents the method from being subclassed
- cannot override a non-private method in a subclass and then make it private
- Data hiding

# Class Diagram revisited



# Class Diagram revisited




```
class Professor {

   private String name;
   private int employeeID;
   private String hireDate;
   private String status;
   private String discipline;
   private float maxLoad;

   public void submitFinalGrade(float grade) {}
   public void acceptCourseOffering() {}
   public void setMaxLoad(float load) {}
   public int takeSabbatical() {}
   public String teachClass() {}
}
```

# Quick Review Questions

- How to define a class?
- How to define an object?
- How to create and call variables?
- How to create and call methods
- Accessors and Mutators
- Access Specifiers
- Class Diagram

# Summary of Main Teaching Points

- Class
- Objects
- Variables
- Methods
- Accessors and Mutators
- Access Specifiers

# Question and Answer Session

A·P·U
ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

Q & A

# Next Session

- **Inheritance**

- Single inheritance
- Multiple inheritance
- Multi-level inheritance
- Inheriting Methods