

CSIT121

Object Oriented Design and Programming

Lesson 2

Real-world objects

- Real-world objects share two characteristics:
 - state and behavior
- Example 1: Book
 - **State** - ISBN: "1-234-345",
 - title: "The End of the Day",
 - author: "Kenny Wilson",
 - publisher: "ABC Publisher"
 - **Behavior** - borrowed, returned, reserved, disposed

Real-world objects

- Example 2: Student
 - **State** - Student ID: "1234567",
 - Name: "James Tay",
 - Address: "30 Kim Keat Ave",
 - Course: "Bachelor of Arts"
 - **Behavior** - register subject, change subject, register exam, withdraw from course

Software objects

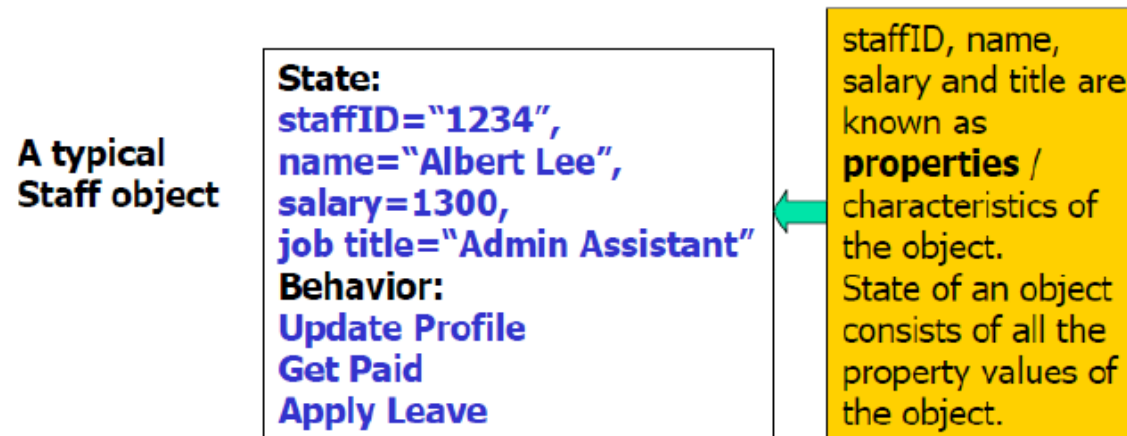
- Software objects are conceptually similar to real world objects
 - consist of state and related behavior.

Real world object	Software object
state	attribute/property
behavior	method

- An object-oriented program is
 - made up of a collection of objects **cooperating** to perform some tasks.
 - Example 1
 - A Library system is possibly made up of objects such a book, member, librarian etc.
 - Example 2
 - An Air Ticket Booking system is possibly made up of objects such as ticket, seat, passenger, airplane etc.

Object

- An object stores its state in variables and exposes its behavior through methods.
- Methods operate on an object's internal state and serve as the primary mechanism for object-to-object communication.



Encapsulation

- Putting all related attributes and methods that operate on the attributes, into a single unit (i.e. a class).
- Benefits
 - Hiding data: Users will have no idea how classes are being implemented or stored. All that users will know is that values are being passed and initialized.
 - More flexibility: Enables you to set variables as read or write-only. Examples include: setName(), setAge() or to set variables as write-only then you only need to omit the get methods like getName(), getAge() etc.
 - Easy to reuse: it's easy to change and adapt to new requirements

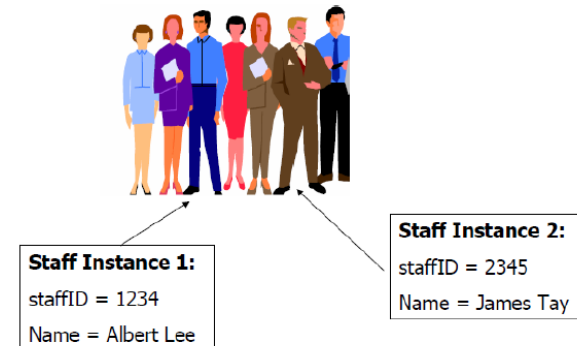
Class

- It is common that objects have the same properties (types of states) and behaviors.
- A company may have hundreds of staffs and all staffs have the same properties and behaviors.
- Class can be created for objects share the same properties and behaviors.



Class

- A class is a blueprint (template) of an object.
 - It establishes the kind of data an object of that type will hold and defines the methods that represent the behaviors of such objects.
- Once a class is defined, multiple objects can be created from that class.
 - Each object is called an INSTANCE of the class.
 - To represent 200 staff, you need to create 200 distinct “Staff” objects or instances.
 - Each instance has its own state.



Class Declaration

- Every class can contain
 - attribute declarations and
 - method declarations.
- Attributes declarations
 - define the kind of data an object of the class can hold
- Method declarations
 - define the function an object has.
 - Usually operate on the attributes
- Each new class created becomes a **new data type** in Java that can be used to declare variables and create objects

Student Class Example

```
1 public class Student{
2     // data declarations
3     private String name;
4     private int mark;
5     // constructor method always should be the first method
6     public Student(String studName, int studMark) {
7         name = studName;
8         mark = studMark;
9     }
10    public String getName () {
11        return name;
12    }
13 }
```

```
14    public void setName (String studName) {
15        name = studName;
16    }
17    public int getMark () {
18        return mark;
19    }
20    public void setMark (int studMark) {
21        mark = studMark;
22    }
23    public String grade ( ) {
24        ...
25    }
26 } //end of class Student
```

Student Class Example

1. Access modifier

- Can be public or private
- For now we will simply declare every class public which means it can be used by any other class.
- Private class will be introduced when discussing GUIs.

2. Class name

- A unique name to identify the class within a package
- Naming convention: Uppercase the first letter of each word

```
1 public class Student{  
    // data declarations  
    private String name;  
    private int mark;
```

Student Class Example

```
1 public class Student{  
2     // data declarations  
3     private String name;  
    private int mark;
```

3. Attribute declarations

- name and mark are called instance variables as memory
- Each student object has its own name and mark values.
- Each instance variable has access modifier associated – either public or private
- If declared with “public” access modifier, it can be referenced from outside of the object.
- If declared with “private” access modifier, it can only be referenced anywhere within the class.
- Generally, instance variables are “private” to ensure data encapsulation.

Student Class Example

4

```
// constructor method always should be the first method
public Student(String studName, int studMark) {
    name = studName;
    mark = studMark;
}
```

4. Constructor

- A special method that initializes a new instance of the class when the object is created using the “new” keyword
- A class needs not provide any constructor. By default, Java compiler provides a default constructor with no parameters for such class.
- Name of constructor must be the same as the class name.
- There is no return type!
- Constructor method is declared as “public”.
- A class may provide one or more constructors with different parameters (method overloading).

Student Class Example


- *this* reference
 - *this* is a special reference in Java and is a reserved word
 - It allows an object to refer to itself within its class definition.
 - *this* reference can be used to distinguish the parameters of a constructor from their corresponding instance variables with the same names.

```
public class Student{  
    // data declarations  
    private String name;  
    private int mark;  
  
    // constructor method always should be first  
    public Student(String name, int mark) {  
        this.name = name;  
        this.mark = mark;  
    }  
}
```

Student Class Example

5. Accessors

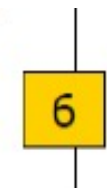
- Methods that provide read-only access to instance variables
- The name is usually get<Attribute Name>.
- E.g. getName, getMark



```
public String getName () {  
    return name;  
}
```

6. Mutators


- Methods that allow changes to values of instance variables
- The name is usually set <Attribute Name>.
- E.g. setName, setMark



```
public void setMark (int studMark) {  
    mark = studMark;  
}
```

Student Class Example

7. Other methods that support the functionalities required for the application
 - You may have as many such methods as required by the specific application.



```
public String grade ( ) {  
    ...  
}
```


Creating objects

- A class provides the blueprint for creating objects:

```
Student aStudent = new Student("Mary",80);
```

- The above statement has three parts
 - **Declaration:** the first part is variable declaration that associates a variable name with an object type.
 - **Instantiation:** The new keyword is a Java operator that creates the object.
 - **Initialization:** The new operator is followed by a call to a constructor, which initializes the new object.
- When an object is created from a class, it has its own distinct copies of instance variables
- In the case of Student objects, instance variables are name and mark

Scope of Instance Variables

- Scope of instance variables (name, mark)– anywhere within the class Student
- Scope of local variables (e.g. studGrade)– anywhere within the method (e.g. grade) that they are declared

```
public class Student{  
    // data declarations  
    1 private String name;  
    private int mark;  
    ...  
  
    public String grade ( ) {  
        String studGrade; 2  
        ...  
    }  
    ...  
}
```

Testing the class... object

- Need to create a new class with the “main” method to test the written class.
- In the main method of this class, create objects of the class being tested and invoke each method (constructor, accessors, mutators and all other methods) to test them.

```
class TestStudent{  
    public static void main (String[] args) {  
  
        // create 3 Student objects/instances  
        Student stud1 = new Student ("Peter", 60);  
1 Student stud2 = new Student ("Kate", 75);  
        Student stud3 = new Student ("John", 90);  
  
        // add 5 marks to student 1  
2 stud1.setMark (stud1.getMark() + 5);  
3 System.out.println(stud1.getName() + "\t" + stud1.getMark());  
    }  
}
```

Testing the class... object

1. Create 3 instances of “Student” class using “new” Keyword
Constructor method is executed

```
class TestStudent{  
    public static void main (String[] args) {  
  
        // create 3 Student objects/instances  
        Student stud1 = new Student ("Peter", 60);  
1 Student stud2 = new Student ("Kate", 75);  
        Student stud3 = new Student ("John", 90);  
  
        // add 5 marks to student 1  
2 stud1.setMark (stud1.getMark() + 5);  
3 System.out.println(stud1.getName() + "\t" + stud1.getMark());  
    }  
}
```

Testing the class... object

2. Call the mutator method `setMark` and assessor method `getMark` of `stud1` instance.
 - Take note that method name is prefixed with the object variable (`stud1`) followed by a dot.
 - `getMark` obtains the current mark and `setMark` sets then new mark to be (current mark + 5).

```
class TestStudent{  
    public static void main (String[] args) {  
  
        // create 3 Student objects/instances  
        Student stud1 = new Student ("Peter", 60);  
1 Student stud2 = new Student ("Kate", 75);  
        Student stud3 = new Student ("John", 90);  
  
        // add 5 marks to student 1  
2 stud1.setMark (stud1.getMark() + 5);  
3 System.out.println(stud1.getName() + "\t" + stud1.getMark());  
    }  
}
```

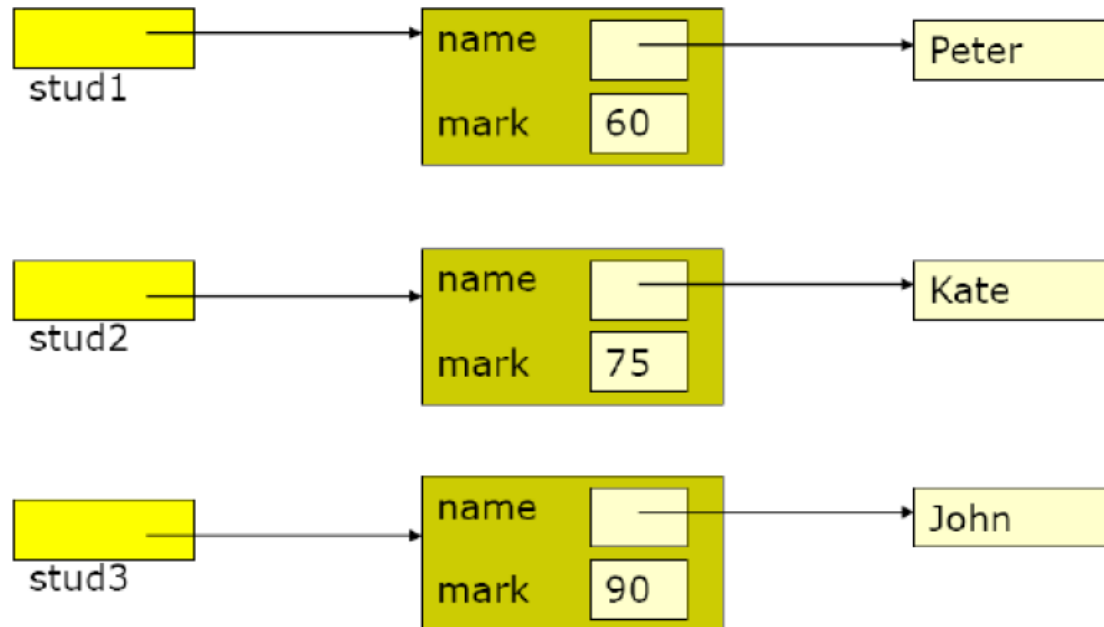
Testing the class... object

3. Call the accessor methods getName and getMark
 - getName and getMark obtain the name and mark respectively

```
class TestStudent{  
    public static void main (String[] args) {  
  
        // create 3 Student objects/instances  
        Student stud1 = new Student ("Peter", 60);  
1      Student stud2 = new Student ("Kate", 75);  
        Student stud3 = new Student ("John", 90);  
  
        // add 5 marks to student 1  
2      stud1.setMark (stud1.getMark() + 5);  
3      System.out.println(stud1.getName() + "\t" + stud1.getMark());  
    }  
}
```

Testing the class... object

- stud1, stud2 and stud3 are **object reference variables**.
- Strings are also objects !



Object Reference Variables

- Variables that are declared as being of class types are “object reference variables”
 - Student stud1;
 - Contain either null or address returned by new operator
- Note: Java does not expose object address, only object ID is shown.

Object toString()

- In Java, every new class **inherits** (later in the course) from `java.lang.Object` class of Java API
- All objects' instances possess a few standard methods.
- For example: `toString()`
 - The string returns by `toString()` represent the state of object.
 - mainly for debugging
 - but can override in your class and define another `toString()` that returns a String representation of data in object
 - `println()` on instance of class invokes `toString` method

Object toString()

```
public class Student{  
    ...  
    public String toString() {  
        return (name + "\t" + mark);  
    }  
}
```

This **toString**
method will be
called

```
class TestStudent{  
    public static void main (String[] args) {  
        Student stud1 = new Student ("Peter", 60);  
  
        // add 5 marks to student 1  
        stud1.setMark (stud1.getMark() + 5);  
        System.out.println (stud1);  
    }  
}
```

Static variable

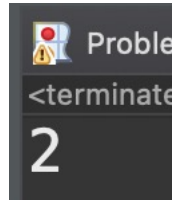
- When a number of objects are created from the same class blueprint, they each have their own distinct copies of instance variables.
- Sometimes, you want to have variables that are common to all objects.
- For example: `studentCount` to keep track of how many `Student` objects have been created.
- This is accomplished with the static variables.
- Static variable are associated with the class, rather than with any object.
- Every instance of the class shares a class variable.

Static variable

- Also known as class variables
- Unlike instance variables, there is only one copy of a static variable for ALL objects of the class.

```
public class Student {  
    private static int studentCount=0;  
    private String name;  
    private int mark;  
  
    public Student(String name, int mark) {  
        this.name = name;  
        this.mark = mark;  
        studentCount++;  
    }  
  
    public static int getStudentCount() {  
        return Student.studentCount;  
    }  
}
```

```
public static void main(String[] args) {  
    Student stud1 = new Student("Peter",70);  
    Student stud2 = new Student("John",60);  
    System.out.println(Student.getStudentCount());  
}
```

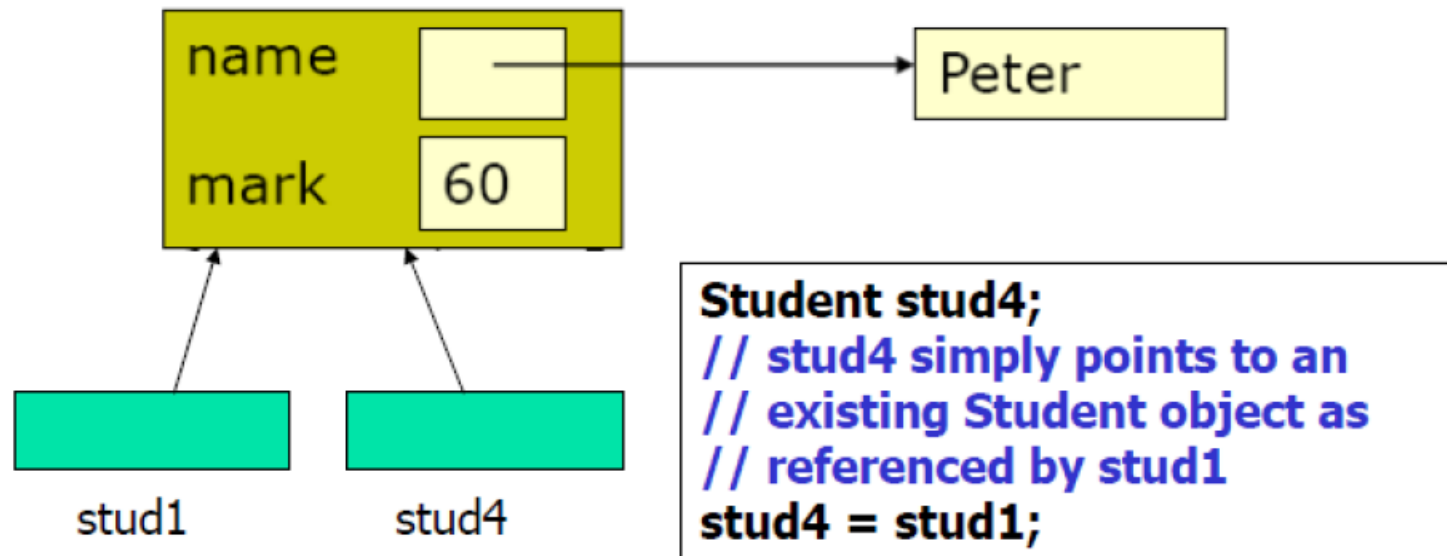


Static Methods

- Also known as class methods.
- Static methods can be invoked through the class name.
- Can only access the static variable of the class (not instance variable).
- Don't have to instantiate an object of the class in order to invoke the method.
- All methods of Math class are static methods.
 - Examples:
 - Integer.parseInt
 - Math.sqrt
- main method of any class.

Object Identity

- There can be multiple references to the same object.



Object Identity

- It means that different instances of same class (even if have identical values in their data fields) are distinguishable.
- Object identity is (in practical terms) synonymous with object address.
- (a == b) tests the addresses held in these variables (i.e. do they refer to the same object?)

```
public static void main(String[] args) {  
    Student stud1 = new Student("Peter",70);  
    Student stud2 = new Student("Peter",70);  
    System.out.println(stud1==stud2); //false  
    Student stud3 = stud2;  
    System.out.println(stud3==stud1); //false  
    System.out.println(stud3==stud2); //true  
}
```

Object Equality

- If you want to check whether two objects (instances of the same class) are “equal”, you have to provide an equals method in your class
- This method would compare the values of data members

```
public boolean equals(Student student) {
    boolean equals = false;
    if(this.name.equals(student.getName())) {
        if(this.mark == student.getMark()) {
            equals = true;
        }
    }
    return equals;
}

public static void main(String[] args) {
    Student stud1 = new Student("Peter",70);
    Student stud2 = new Student("Peter",70);
    Student stud3 = new Student("John",70);
    System.out.println(stud1.equals(stud2)); //true
    System.out.println(stud1.equals(stud3)); //false
}
```


ArrayList

- Disadvantage of array
 - Size of array is fixed once it is declared.
 - No way to increase the size during the duration of program execution
- ArrayList class in java is designed to overcome the above problem.
- Using ArrayList is similar to array, however the size of ArrayList is “dynamic”.
- The size can grow or shrink while the program is running

ArrayList

- Declare and create an ArrayList
- Add an object

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add("Ada");
```

```
names.add("Bob");
```

ArrayList

- Get an object using index

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add("Ada");
```

```
names.add("Bob");
```

```
System.out.println(names.get(0));
```

```
System.out.println(names.get(1));
```

ArrayList

- Loop an ArrayList

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add("Ada");
```

```
names.add("Bob");
```

```
//using for loop
```

```
for(int i=0;i<names.size();i++) {  
    System.out.println(names.get(i));  
}
```

```
//using for each loop
```

```
for(String name: names) {  
    System.out.println(name);  
}
```

ArrayList

- Remove an object

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add("Ada");  
names.add("Bob");  
names.add("Charles");
```

```
//remove Bob  
names.remove(1);  
//Charles is moved to index 1
```

```
//using for loop  
for(int i=0;i<names.size();i++) {  
    System.out.println(names.get(i));  
}
```

ArrayList

- Set an object in the ArrayList at a specified index

```
ArrayList<String> names = new ArrayList<String>();
```

```
names.add("Ada");  
names.add("Bob");  
names.add("Charles");
```

```
//remove Bob  
names.remove(1);  
//Charles is moved to index 1
```

```
//replace Ada with Darren  
names.set(0, "Darren");
```

```
//using for loop  
for(int i=0; i<names.size(); i++) {  
    System.out.println(names.get(i));  
}
```

Array vs ArrayList

	Array	ArrayList
Size	Need to declare in advance how many elements	Size can grow and shrink as required
OO Support	Not implemented as Class.	Implemented as Class.
Efficiency	More efficient then ArrayList	Not as efficient as Array
Storing of data types	Can store both primitive and non-primitive data type	Can only non-primitive data type. To store primitive data type needs to use a wrapper class.

```
ArrayList<Integer> intAL = new ArrayList<Integer>();  
intAL.add(1);  
intAL.add(2);  
intAL.add(3);  
  
ArrayList<Double> doubleAL = new ArrayList<Double>();  
doubleAL.add(1.1);  
doubleAL.add(2.2);  
doubleAL.add(3.4);
```