Classes and Objects

Lecture 2.1

Object Oriented Paradigms

College Requirements - Compulsive Courses

CSCR2105

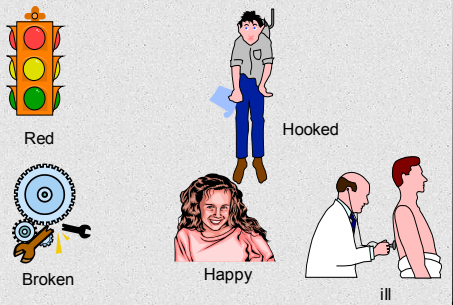
Objects everywhere...

OO Programming Concepts

- Object-oriented programming (OOP) involves programming using objects.
- An object represents an **entity** in the real world that can be distinctly identified.
- An object has a unique identity.
  - The **state** of an object consists of a set of **data fields** (also known as properties) with their **current values**.
  - The **behavior** of an object is defined by a set of **methods**.

Objects have behavior....

Objects have state...



Abstraction

- abstraction: A distancing between ideas and details.
  - Hides (or ignores) unnecessary details
  - We can use objects without knowing how they work.
  - Denotes the essential properties of an object.
  - One of the fundamental ways in which we handle complexity.
  - Objects are abstractions of real world entities
  - Programming goal: choose the right abstractions.

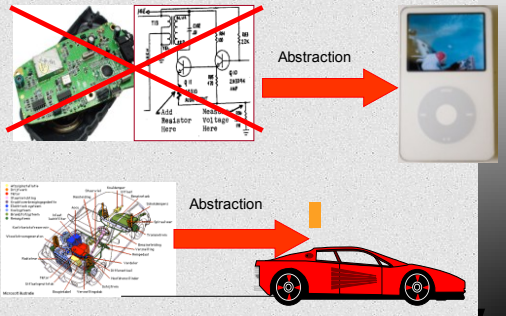
World & Describing the world

- The world is
  - a set of things
  - interacting with each other.
- Describe a particular **person**
  - ***Ahmed*** has long blond hair, green eyes, is 1.63m tall, weighs 56Kg and studies computer engineering. Now lying down asleep.
  - ***Alaa*** studies electronics, has short black hair and brown eyes. He is 180cm and 75 kilos. Now running to class!
- Notice how all have specific values of
  - name, height, weight, eye colour, state, ...

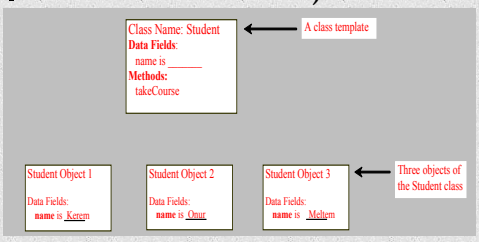
Introduction to Objects

- An *object* represents something with which we can interact in a program
- An object provides a collection of services that we can tell it to perform for us
- The services are defined by methods in a *class* that defines the object
- A class represents a concept, and an object represents the embodiment of a class
- A class can be used to create multiple objects

Abstraction



Classes and objects



An object has both a **state** and **behavior**. The **state** defines the **object**, and the **behavior** defines **what the object does**.

Classes and objects

- **class**: A program entity that represents either:
  1. A program / module, or
  2. A template for a new type of objects.
- A Java class uses
  1. **Variables** to define **data fields** and
  2. **Methods** to define **behaviors**.
- **Object**: An entity that combines **state** and **behavior**.
  - Object-oriented programming (OOP): Programs that perform their behavior as interactions between objects.

Classes and objects

```
public class Student {
      private String name;
      private String studentID;
      private Course[] takenCourses;
      private int state;
      public void takeCourse(Course course) {
            .
            .
            .
      }
      public void Student() {
            .
            .
            .
      }
}
```

Objects and Classes

A class
(the concept)

Bank Account

An object
(the realization)

Ahmed's Bank Account
Balance: $5,257

Omer's Bank Account
Balance: $1,245,069

Ali's Bank Account
Balance: $16,833

Multiple objects
from the same class

Constructors

- A class provides a special type of methods, known as constructors, which are invoked to construct objects from the class.
  - A constructor with no parameters is referred to as a no-arg constructor.
  - Constructors must have the same name as the class itself.
  - Constructors do not have a return type—void.
  - Constructors are invoked using the new operator when an object is created.
  - Constructors play the role of initializing objects.

16

```
public class BankAccount{
      private String name;
      private double balance;

 public void initialize(){
            balance = 5,257;        //usage
            name = " Ahmed's ";     //usage
      }
 public void display () {
            System.out.println("Name " + name);
            System.out.println(" Balance $"+ Balance);
      }
}
class ExBankAccount{
      public static void main(String [] args) {
            BankAccount a = new BankAccount();  //creation
            a.initialize();                     //usage
            a.display();                        //usage
            }
   }
```

```
class BankAccount
{
    private String name;
    private double balance ;
  BankAccount() // default constructor. No argument list
    {
            name = "Ali ";
            balance = 1,000;
      }
    public BankAccount(String n, double b) //non-default constructor
    {
            name = n;
           balance = b;
    }
    void display()
    {
            System.out.println("\n Bank Account Info ");
            System.out.println("Name "+ name);
            System.out.println("Balance $"+ balance );
    }
}
```

Constructors

- A class may be declared without constructors.
  - In this case, a no-arg constructor with an empty body is implicitly declared in the class.
  - This constructor, called a default constructor, is provided automatically only if no constructors are explicitly declared in the class.

```
BankAccount( ) {
}
BankAccount(String newname, double newbalance )
 {
     balance = newbalance ;
        name = newname;
}
```

Accessing Objects

- Referencing the object's data:

  `objectRefVar.data`

  e.g., `b.balance`
- Invoking the object's method:

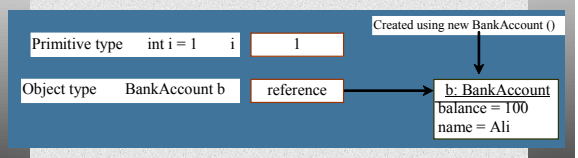
  `objectRefVar.methodName(arguments)`

  e.g., `b.display()`

Creating Objects

1. Creating Objects Using Constructors
   - `new ClassName();`
   - Example:
     - `new BankAccount ();`
     - `new BankAccount ("Amal",10000);`
2. Declaring/Creating Objects in a Single Step
   - `ClassName objectRefVar = new ClassName();`
   - Example:
     - `BankAccount b = new BankAccount ();`

Differences between Variables of Primitive Data Types and Object Types



22

```
class ExBankAccount
{
    public static void main (String [] args)
    {
        BankAccount b1 = new BankAccount();
        b1.display();
        BankAccount b2 = new BankAccount("Ahmed",400);
        b2.display();
        }
}
```

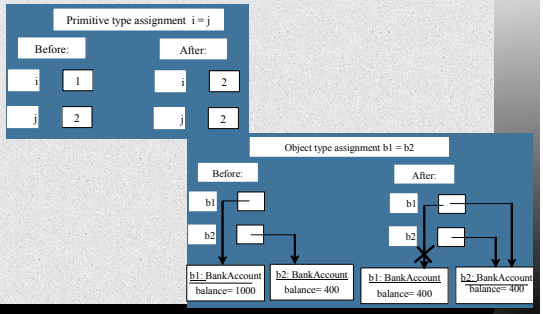
21

Destroying Objects

- No way to explicitly destroy an object
- Objects destroyed by the Garbage Collector
  - Once they go out of scope (I.e. no longer referenced by any variable)
- No way to reclaim memory, entirely under control of JVM
  - There is a `finalize` method, but its not guaranteed to be called (so pretty useless!)
  - Can request that the Garbage Collector can run, buts its free to ignore you

24

Copying Variables of Primitive Data Types and Object Types



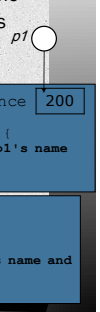
BankAccount objects and method

- Each `BankAccount` object has its own copy of the `display` method, which operates on that object's state:

```
BankAccount p1 = new BankAccount();
p1.name = Omer;
p1. balance = 200;

BankAccount p2 =
new BankAccount();
p2.name = Ali;
p2. balance = 300;

p1.dispaly();
p2.display();
```



Garbage Collection

- As shown in the previous figure,
  - after the assignment statement b1 = b2,
  - b1 points to the same object referenced by b2.
  - The object previously referenced by b1 is no longer referenced.
  - This object is known as garbage. Garbage is automatically collected by JVM.

25

Kinds of class's methods

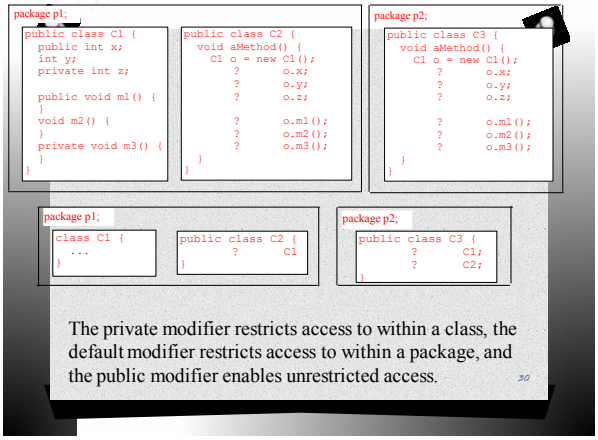
- **accessor**:A method that lets clients examine object state.
  - Examples: `display,getName()`
- **mutator**: A method that modifies an object's state.
  - Examples: `setSame(),Salary()`
  - often has a **non-void** return type

28

The implicit parameter

- **implicit parameter**:
  The object on which an instance method is called.
  - During the call `p1.dispaly();`
    the object referred to by `p1` is the implicit parameter.
  - During the call `p2.dispaly();`
    the object referred to by `p2` is the implicit parameter.
  - The instance method can refer to that object's fields.
    - We say that it executes in the *context* of a particular object.
    - Display can refer to the name and balance of the object it was called on.

27



The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

30

Visibility Modifiers

`public`

The class, data, or method is visible to any class in any package.

`private`

- The data or methods can be accessed only by the declaring class.
- The ***get*** and ***set*** methods are used to read and modify private properties.
- Why Data Fields Should Be private?
  - To protect data
  - To make class easy to maintain

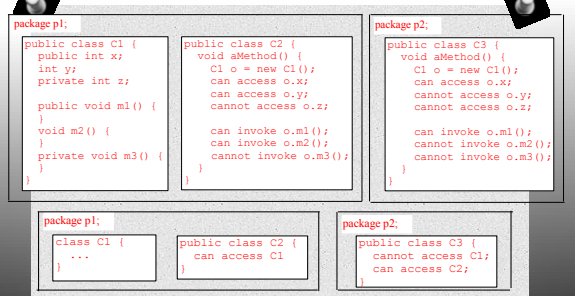
By default, the class, variable, or method can be aaccessed by any class in the same package.

29

Static

- `static` – indicates a *class variable* or *class method* not owned by an individual object
  - This means we don't have to create an object to use it.
  - `Arrays.sort` and `System.arrayCopy` are static methods
- **Static variables** are shared by all the instances of the class.
- **Static methods** are not tied to a specific object.
- **Static constants** are final variables shared by all the instances of the class.

32



The private modifier restricts access to within a class, the default modifier restricts access to within a package, and the public modifier enables unrestricted access.

31

Final

- `final` – to make a variable that can have a single value
  - Can be assigned to *once and once only*
  - Useful to ensure a variable isn't changed once its assigned.

```
final int count;
count = 10;
//the following will cause an error
count = 20;
```

34

Static -- Example

```
public class MyClass
{
  public static void utilityMethod() { … }
  public void otherMethod() { … }
}
//using the above:
MyClass.utilityMethod();
MyClass objectOfMyClass = new MyClass();
objectOfMyClass.otherMethod();
objectOfMyClass.utilityMethod();
//this is illegal:
MyClass.otherMethod();
```

33

What Class is Immutable?

- A class with all private data fields and without mutators is not necessarily immutable.
- For example, the following class Student has all private data fields and no mutators, but it is mutable.
- For a class to be immutable, it must mark all data fields private and provide no mutator methods and no accessor methods that would return a reference to a mutable data field object.

36

Defining Constants

- Unlike other languages, Java has no const keyword
- Must use a combination of modifiers to make a constant
  - **static** – to indicate its owned by the class
  - **final** – to make sure it can't be changed (and initialise it when its declared)
- Naming convention for constants is to use all capitals.

35

```
public class BirthDate {
  private int year;
  private int month;
  private int day;

  public BirthDate(int newYear, int newMonth, int newDay)
  {
          year = newYear;
          month = newMonth;
          day = newDay;
  }
   public void setYear(int newYear)
 {
          year = newYear;
  }
}
```

```
public class Student {
  private int id;
  private BirthDate birthDate;
  public Student(int ssn, int year, int month, int day)
   {
          id = ssn;
          birthDate = new BirthDate(year,month, day);
  }
  public int getId() {
          return id;
  }
  public BirthDate getBirthDate() {
          return birthDate;
  }
}
```

Passing Objects to Methods

- Passing by value for primitive type value (the value is passed to the parameter)
- Passing by value for reference type value (the value is the reference to the object)

40

```
public class Test {
  public static void main(String[] args) {
    Student student = new Student(111223333, 1970, 5, 3);
    BirthDate date = student.getBirthDate();
    date.setYear(2010); // Now the student birth year is changed!
  }
}
```

41

```
public class TestPassObject {
 /** Main method */
        public static void main(String[] args) {
                Circle3 myCircle = new Circle3(1); // Create a Circle object with radius 1
                // Print areas for radius 1, 2, 3, 4, and 5.
                 int n = 5;
                printAreas(myCircle, n);

                // See myCircle.radius and times
                 System.out.println("\n" + "Radius is " + myCircle.getRadius());
                System.out.println("n is " + n);
        }
        /** Print a table of areas for radius */
        public static void printAreas(Circle3 c, int times) {
                System.out.println("Radius \t\tArea");
                 while (times >= 1) {
                        System.out.println(c.getRadius() + "\t\t" + c.getArea());
                        c.setRadius(c.getRadius() + 1);
                         times--;
                }
        }
}
```

```
public class Circle3 {
        private double radius = 1;  // The radius of the circle
        public Circle3() {              // Construct a circle with radius 1
        }
        public Circle3(double newRadius) {  // Construct a circle with a specified radius
                radius = newRadius;
        }
        public double getRadius() {                     // Return radius
                return radius;
        }
        public void setRadius(double newRadius) {       // Set a new radius
                radius = (newRadius >= 0) ? newRadius : 0;
        }
        public double getArea() {               // Return the area of this circle
                return radius * radius * Math.PI;
        }
}
```

Waiting for your questions and comments

NOW:

Lecture 2.2

Classes.