Java Programming
Objects

Arash Habibi Lashkari
Ph.D. Candidate of UTM University
Kuala Lumpur, Malaysia

Objects

- **Object** is the most important concept in OO.
- When OO program is executed, a space is created to hold objects that are created during the program execution
- Objects **interact** with each other within the space

Objects and Classes

- Classes – the **template** or blueprint from which the object is actually made
- Objects – **instances** of a class
- An OO program contains definition of classes, and object instances will be created **(instantiation)** when this program is executed

Definition of Object

- An object has **state**, **behaviour** and **identity**.
- Examples:
  - The LCD projector in FTSM's lecture hall.
  - Dean's Perdana
  - ATM machine in FKej

Object State

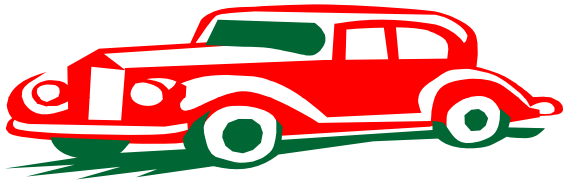
- Each object has attributes which collectively represent its state.
- Example:

state

Attributes :
Colour : black
Plate No : WNH8888
Owner : Ali

Object State

- Imagine two cars :



Attributes :
Colour : black
Plate No : WNH8888
Owner : Ali

Attributes :
Colour : red
Plate No : WNP6666
Owner : Ali

- The first car has a different state from the second car

Object State

- The attributes of an object normally do not change.
- It is the value of the attributes of an object that can change.

Attributes :
Colour : red
Plate No : WNP6666
Owner : Ali

Object State

- The attributes of an object normally do not change.
- It is the value of the attributes of an object that can change.

Attributes :
Colour : yellow
Plate No : WNP6666
Owner : Ali

Object Identity

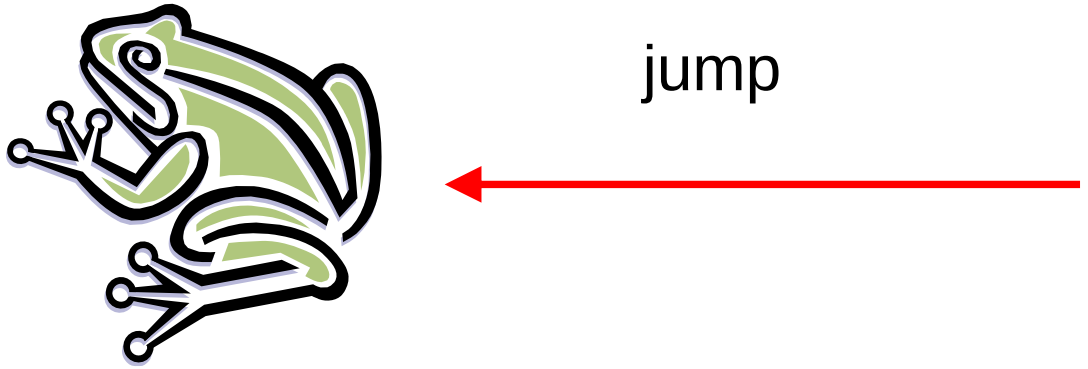
- Each object has its own **identity** which differentiates it from other objects.

Triplets… but each of them has its own identity

Object Behaviour

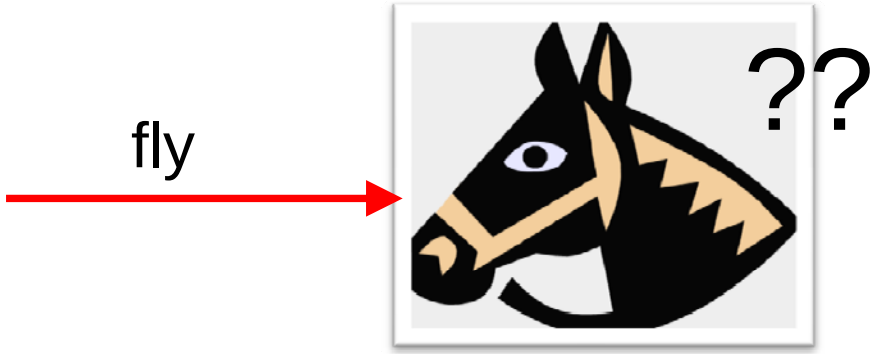
- **Object behaviour** refers to how an object reacts to messages it received

  → determines the actions and roles of an object
- A **message** is sent to an object to initiate the object to perform a particular action.



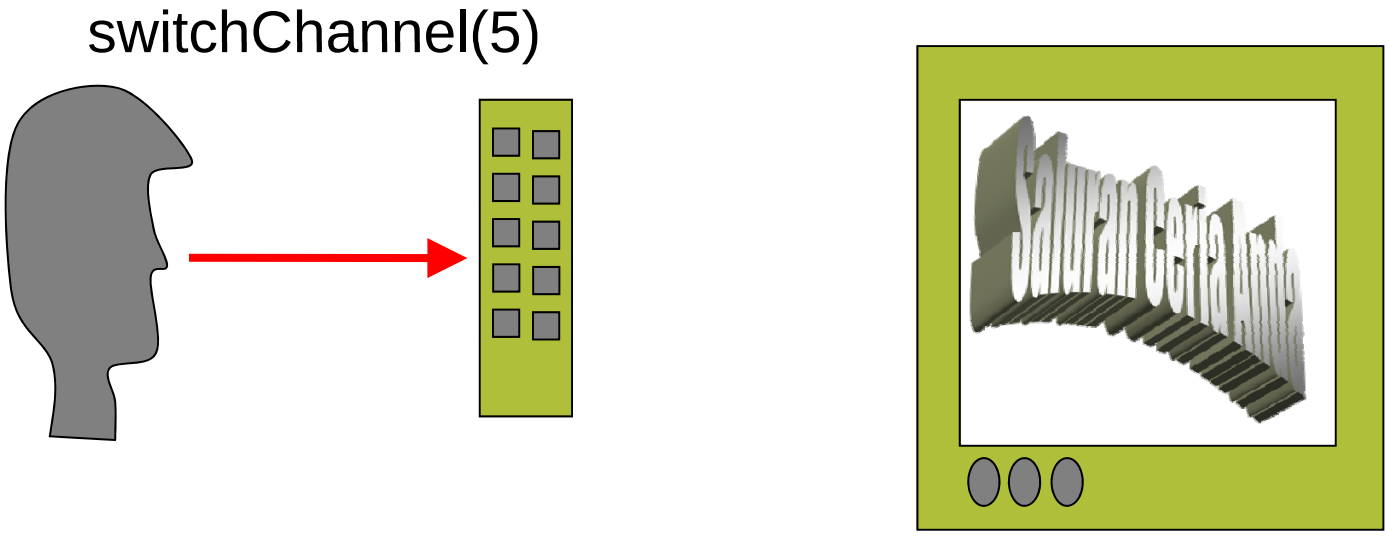
Object Behaviour

- **Message sending** is a mechanism which objects use to interact with other objects.
- An object only responds to messages that it understands.



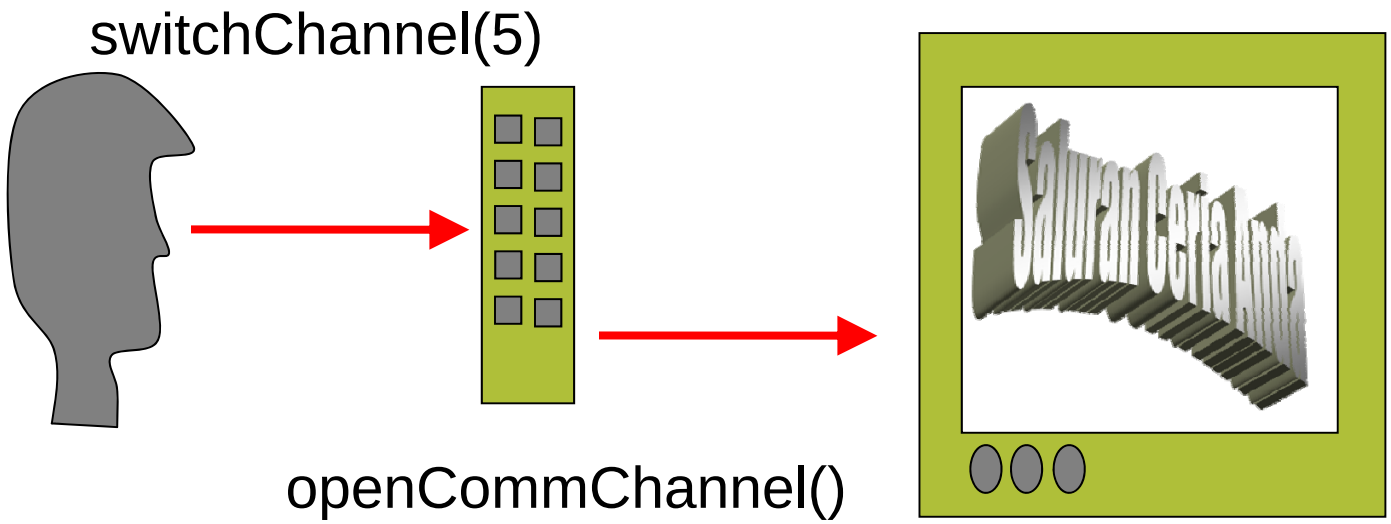
Object Behaviour

- An object may also send messages to other objects in its response to a message.



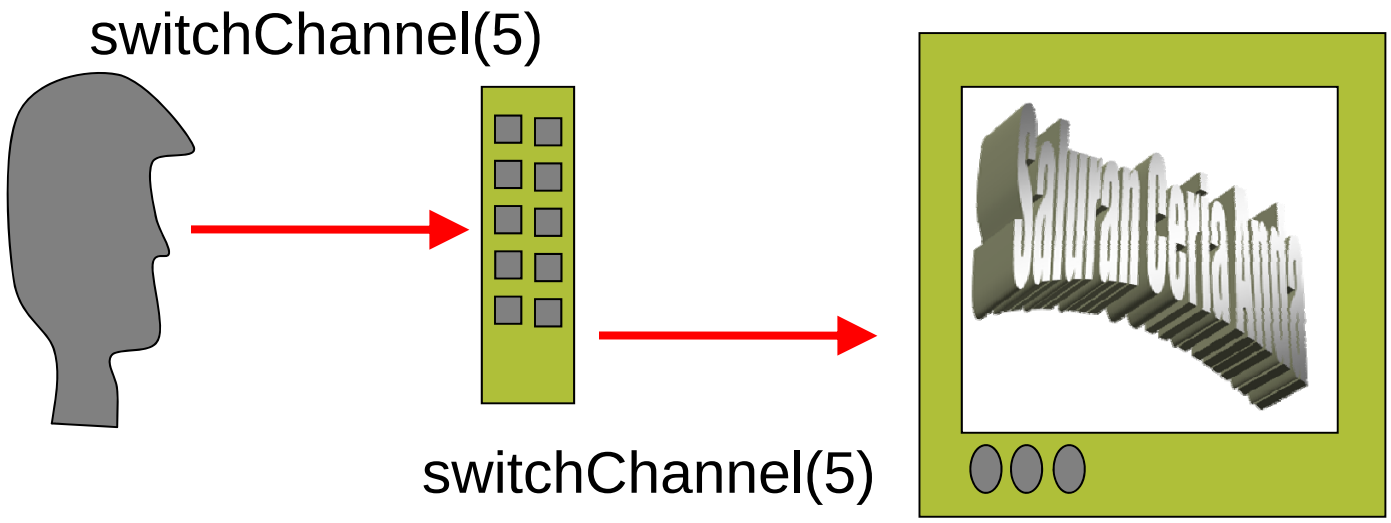
Object Behaviour

- An object may also send messages to other objects in its response to a message.



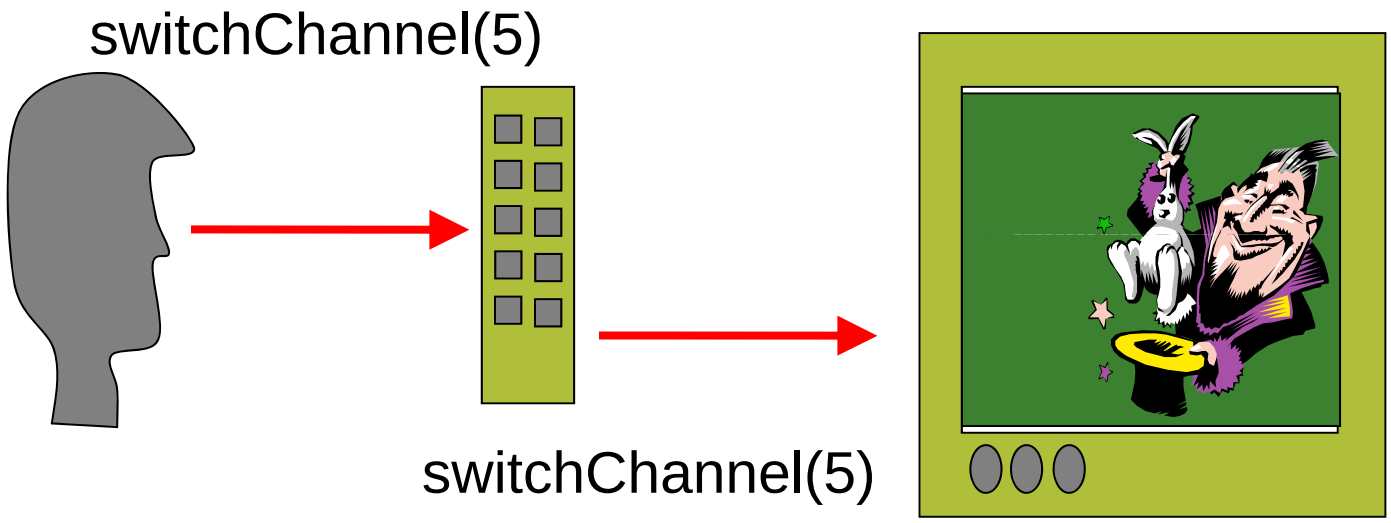
Object Behaviour

- An object may also send messages to other objects in its response to a message.



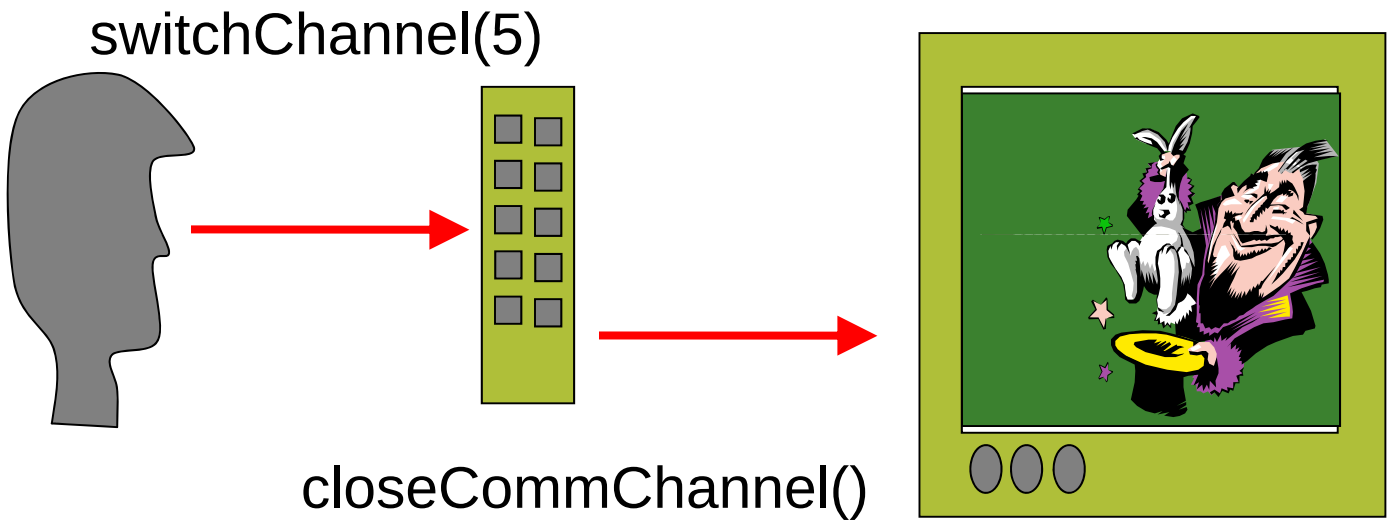
Object Behaviour

- An object may also send messages to other objects in its response to a message.



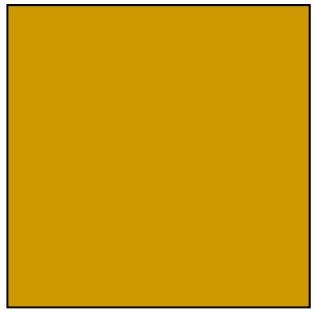
Object Behaviour

- An object may also send messages to other objects in its response to a message.



Object Behaviour

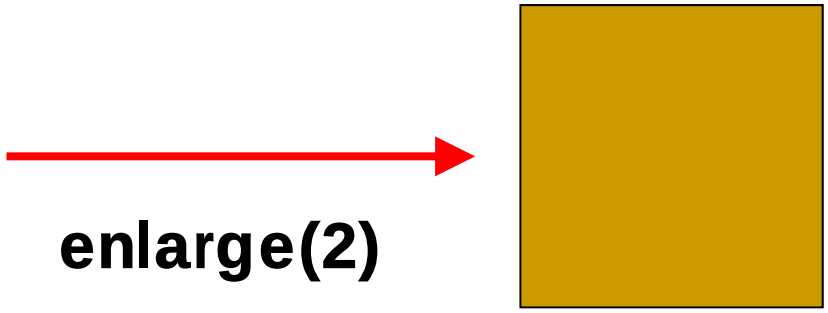
- An object's response to a message can cause its state to change.



Original state :
width = 2
length = 2

Object Behaviour

- An object's response to a message can cause its state to change.



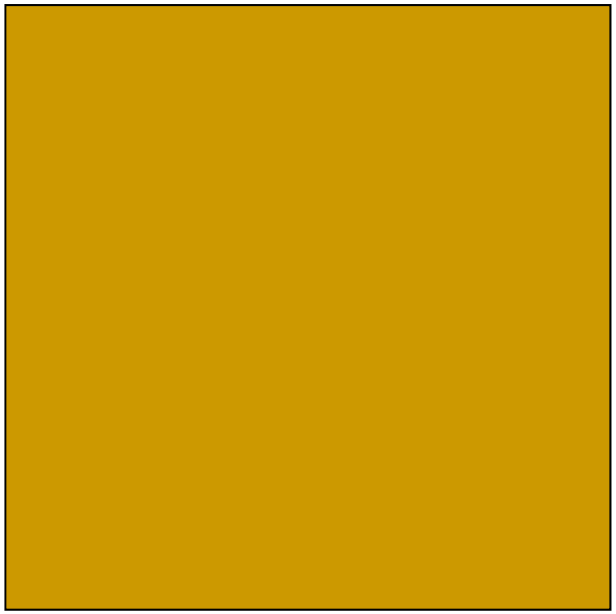
Original state :
width = 2
length = 2

Object Behaviour

- An object's response to a message can cause its state to change.

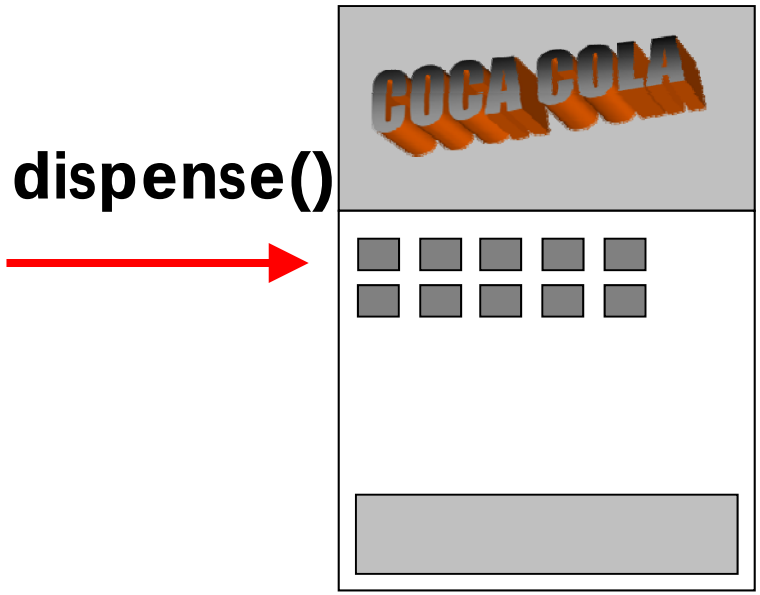
enlarge(2)

<u>New state</u> :
width = 4
length = 4



Object Behaviour

- Changes in object state may change the object behaviour

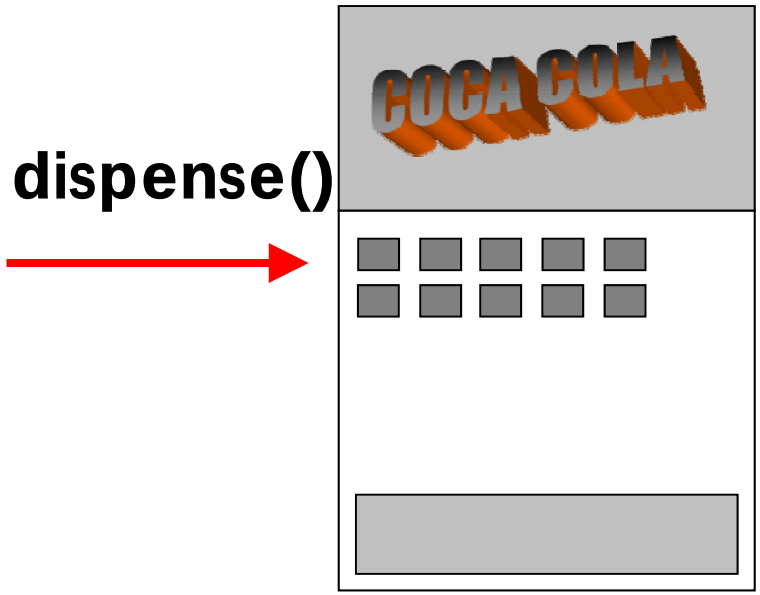


Original Behaviour :
dispense a can

Original State:
Number of cans = 10

Object Behaviour

- Changes in object state may change the object behaviour

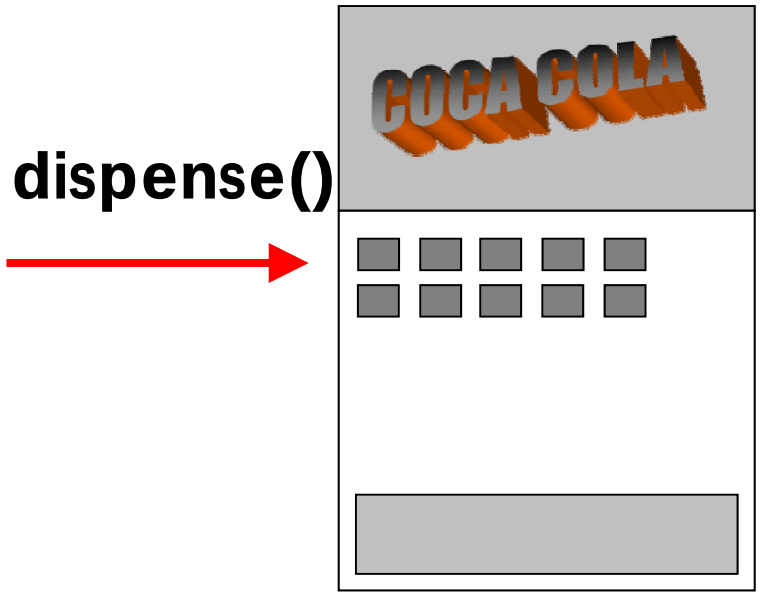


Original Behaviour :
dispense a can

New State:
Number of cans = 9

Object Behaviour

- Changes in object state may change the object behaviour

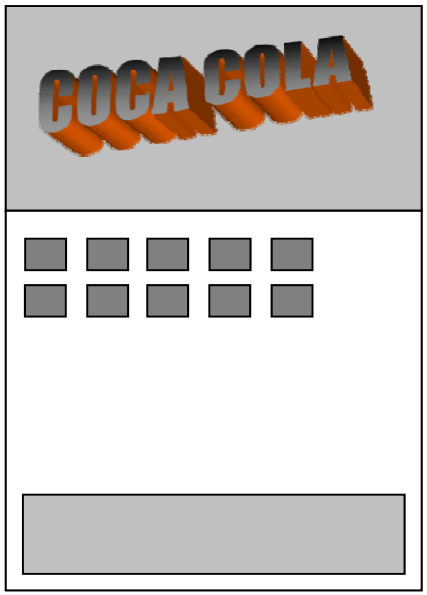


Original Behaviour :
dispense a can

New State:
Number of cans = 0

Object Behaviour

- Changes in object state may change the object behaviour



New Behaviour :
Stop dispensing a can, alert to add more cans

New State:
Number of cans = 0

Object Behaviour

- Object behaviour can affect object state.
- Object state can affect object behaviour.

Object Collaboration

- Isolated objects are useless in object-oriented systems.

"No object is an island"

Implementation

Object State

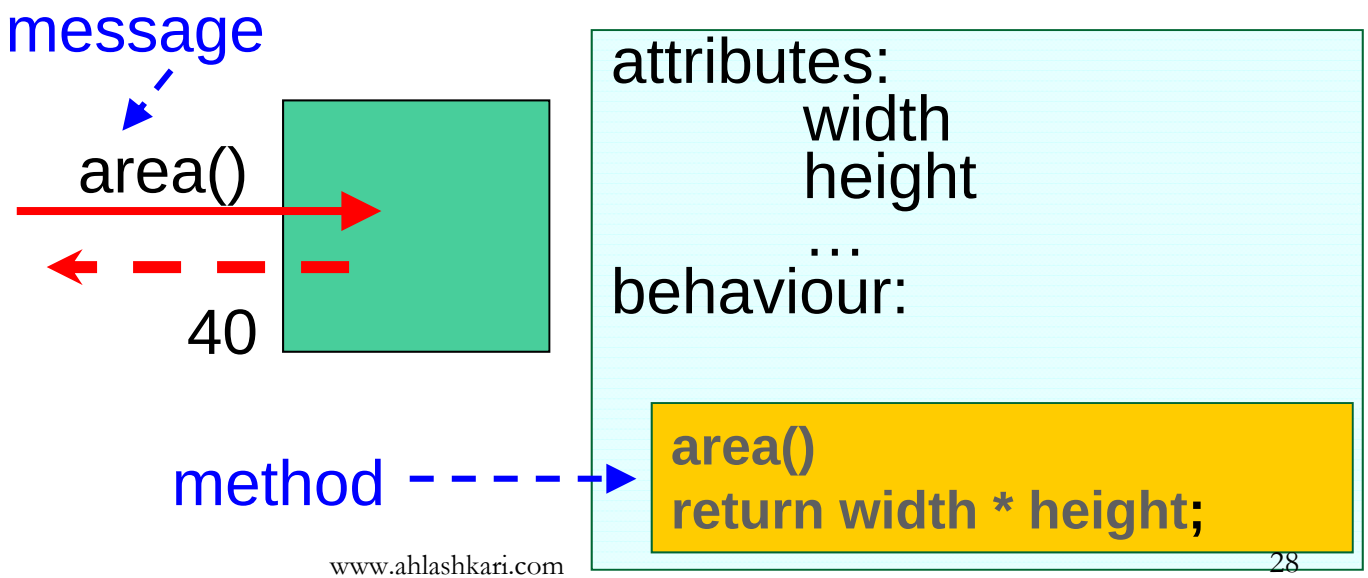
- In Java, the attributes of an object are implemented as variables that belong to that object → **instance variables**.
- Example:

CAR
Colour : black
PlateNo : WNH8888
Owner : Ali

CAR
Colour : red
PlateNo : WNP6666
Owner : Ali

Object Behaviour

- An object implements its behaviour with methods.
- A method contains implementation details of how an object responds to a certain message.



Using Existing Classes

Using Existing Classes

- Create an object (and specify its initial state)
- Send messages to the object
- Access object attributes
- Destruct an object

Object Creation in Java

- The **new** Java keyword is for creating objects.
- To create a new object of a particular class:

  new <*className*>(<*parameterList*>)
- This depends on the constructor defined in the class
- Constructor – a method which has the same name as the class

Object Creation in Java

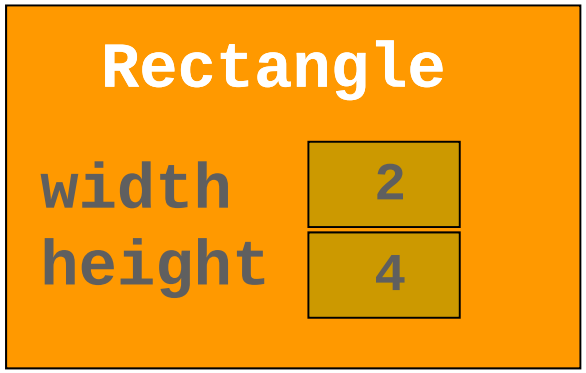
- When an object is created, sufficient amount of memory will be allocated for storing its state.

Object Creation in Java

- Example :

  new Rectangle(2, 4)

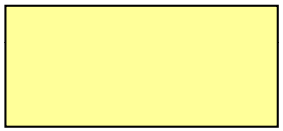
  create a **Rectangle** object whose width and height are 2 and 4 respectively



Object Variables

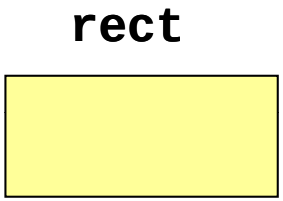
Rectangle rect;

rect

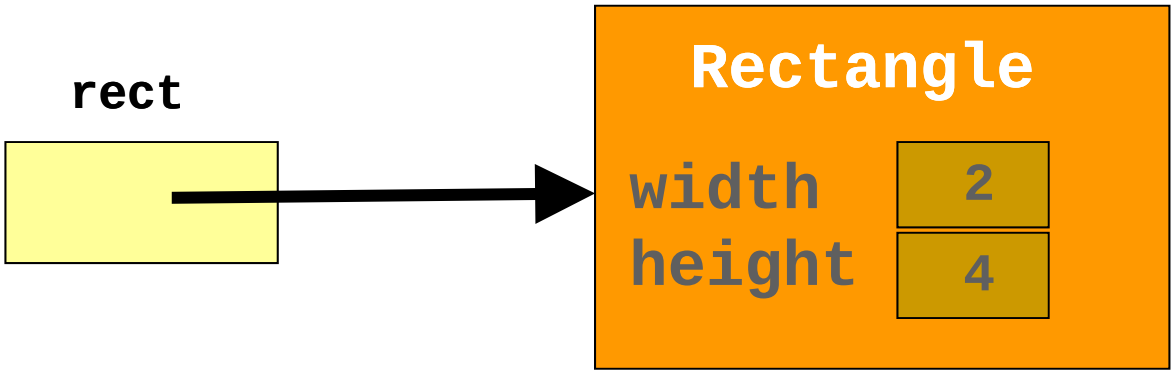


Object Variables

Rectangle rect;



rect = new Rectangle(2, 4);



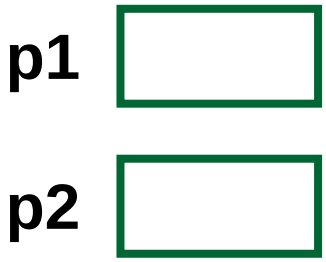
Object Variables

Consider the following Java code:

```
Person p1, p2;
p1 = new Person(“ARA");
p2 = new Person(“AHL");
p1 = new Person(“ARAAHL");
```

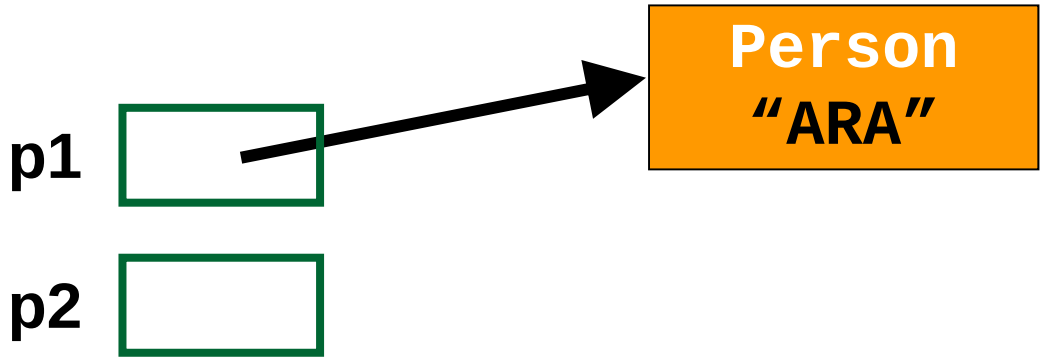
Object Variables

Person p1, p2;



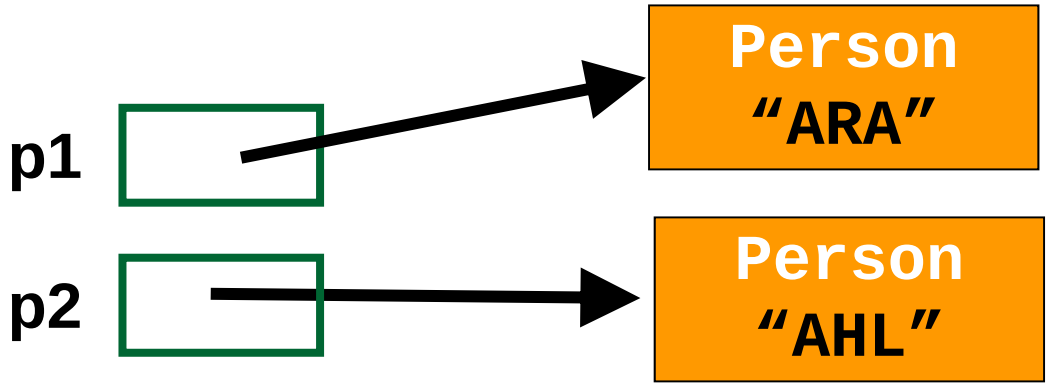
Object Variables

```
p1 = new Person("Azizi");
```



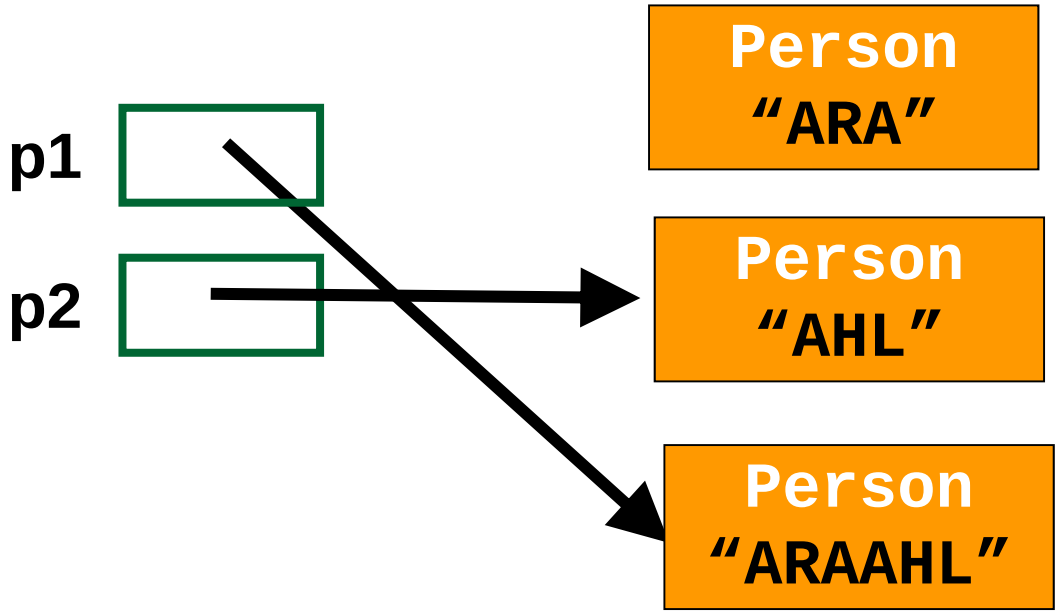
Object Variables

p2 = new Person("Belal");



Object Variables

```
p1 = new Person("Ismail");
```



Sending a message

- To send a message to a particular object, use the **. operator**
- Syntax:

  <object>.<msg_name>(<par_list>)
- Example:

```
console.nextInt()
game1.play()
rect.area(3, 5)
(new Rectangle()).area(3,5)
```

Message and Methods

- A message sent to an object must match one of the methods defined for the object
  - The **name** of the message is the same as the name of the method
  - The number and type of **parameters** sent must be the same as the parameters defined in the method (if any)

Using Predefined Methods

- Predefined methods are methods defined in predefined classes
- Example of predefined classes:
  - **Math**
  - **Integer**
  - **Scanner**
- To use predefined methods, the package where the related class is defined must be imported first
- Example:

```
import java.util.*;
```

User-defined Methods

- Can be classified into two categories:
  - **Value-returning methods** – have a return data type, and return the value using **return** statement
  - **Void methods** – do not have a return data type, do not use a **return** statement

User-defined Methods

- Definition :

  public int getArea(int x, int y)

- Message sending:

  area = rect.getArea(2, 4);

  or

  System.out.println("Area" + rect.getArea(2, 4));

  or

  total = circleArea + rect.getArea(2, 4);

  or

  area = (new Rectangle()).getArea(2, 4);

Accessing Object Attributes

- The dot operator is also used for accessing object attributes.
- Syntax:

  <object>.<attributeName>

- Example :

  rect.width

  rect.height

Object Destruction

- The Java runtime system provides support for **garbage collection**.
- The garbage collector is a background process responsible for finding garbage in memory and **automatically** freeing it.
- Java programmers are no longer burdened with the responsibility of freeing memory

Example of a class definition

```
public class Clock {
    // instance variables
    private int hr;
    public int min;
    private int sec;

    public Clock()
    {
        // default constructor
    }
```

Example of a class definition

```
    public void setTime( int hours, int minutes, int
    seconds)  {
        // method to set the time
    }

    public int getHours() {
        // method to return the hours
    }
    :
}
```

Example of program using Class Clock

```
public class TestClock {
    public static void main (String[] args) {
        Clock myClock = new Clock();
        System.out.println("Original hour : " +
        myClock.getHours());
        System.out.println("Original minute : " +
        myClock.min);
        myClock.setTime(12, 30, 15);
        System.out.println("New hour : " +
        myClock.getHours());
    }
}
```

Objects (Part 2)

Abstraction

- Class abstraction – separation of the class implementation from the use of a class
- The clients can use the class without knowing the implementation of the class

  → only through the description of the class

Abstraction

- Analogy :

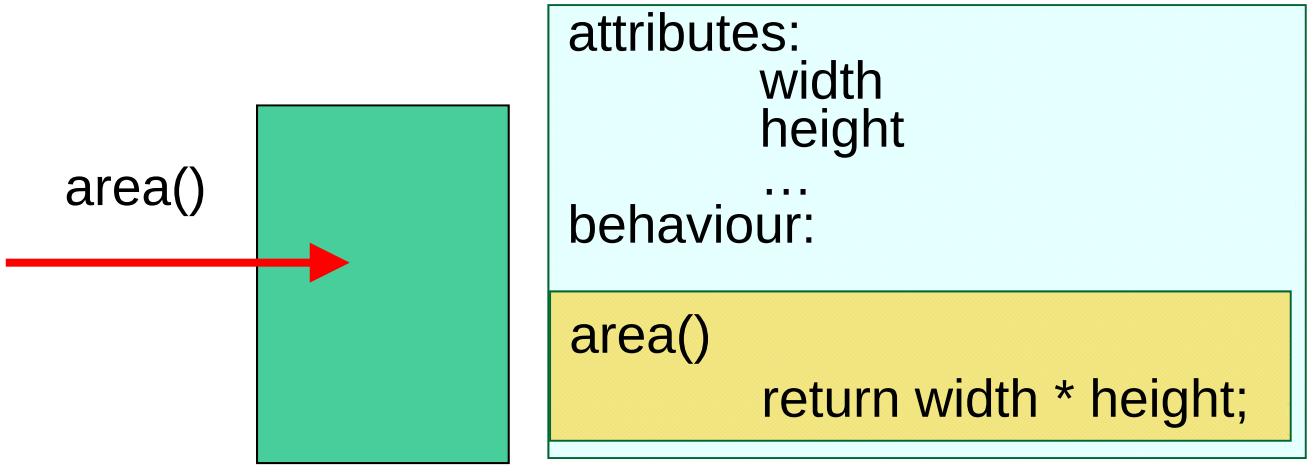
  A restaurant may not want to reveal the specifics of how the dishes it serves are prepared. On the other hand, it needs to reveal the price of each type of dish.

Encapsulation

- The details of the implementation are encapsulated and hidden from the user
- Information hiding or encapsulation is the technique for packaging information in such a way as to hide what should be hidden, and make visible what is intended to be visible.

Encapsulation

- Information normally hidden by objects:
  - object attributes
  - the specifics of how they react to the messages received by them



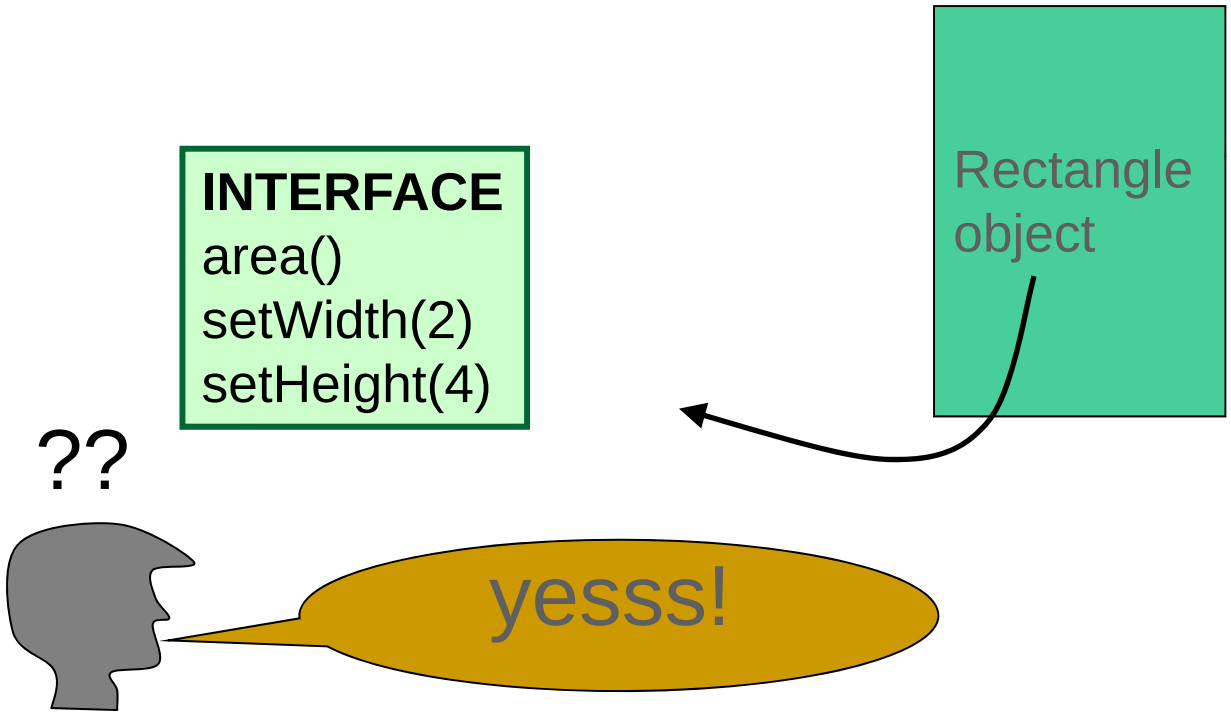
Encapsulation

- Some benefits of encapsulation:
  - if done properly, modifications to object implementation should not affect clients
  - objects can control access to their attributes

Object Interface

- The description of the class given to enable its use
- The interface of an object reveals **public information** about the object.
- An example of information that could be in an object's interface is a list of its **public methods**; only their signature not their implementation.

Object Interface



Example 1 : The Course Class

Course
-name: String -students: String[] -numberOfStudents: int
+Course(name: String) +getName() : String +addStudent(student: String): void +getStudents(): String[] +getNumberOfStudents(): int

Example 1 : The Course Class

Write a program that :

- Creates 2 courses : C++ and Java
- Adds the following students to the C++ course:
  - Syamil Anwar
  - Toh Kim Lian
  - Sarasa Pasamanickam
- Adds the following students to the Java course:
  - Syamil Anwar
  - Toh Kim Lian
- Prints out the number of students taking the C++ and the list of their names
- Prints out the number of students taking the Java course

Example 1 : The Course Class

```
public class TestCourse {
    public static void main(String[] args) {
        Course course1 = new Course("C++");
        Course course2 = new Course("Java");

        course1.addStudent("Syamil Anwar");
        course1.addStudent("Toh Kim Lian");
        course1.addStudent("Sarasa Pasamanickam");

        course2.addStudent("Syamil Anwar");
        course2.addStudent("Toh Kim Lian");
    }
}
```

Example 1 : The Course Class

```
        System.out.println("No of students in course1: " +
        course1.getNumberOfStudents() );

        String[] students = course1.getStudents();
        for (int i = 0; i < course1.getNumberOfStudents(); i++)
                System.out.print(students[i] + ", ");

        System.out.println();
        System.out.println("Number of students in course2: " +
        course2.getNumberOfStudents() );
    }
}
```

Example 2: The Clock Class

Clock
-hr: int -min: int -sec: int
+Clock() +Clock(hours: int, minutes: int, seconds:int) +setTime() : void +getHours(): int +getMinutes(): int +getSeconds(): int +printTime(): void +incrementSeconds(): void +equals(otherClock: Clock) : boolean +makeCopy(otherClock: Clock): void +getCopy(): Clock

Example 2: The Clock Class

```
public static void main (String[] args) {
        Clock defaultClock = new Clock();
        Clock nextClock = new Clock();
        Clock futureClock = newClock();
        int h, m, s;
        Scanner cn = new Scanner(System.in);
        h = cn.nextInt();
        m = cn.nextInt();
        n = cn.nextInt();
        Clock currentClock = new Clock(h, m, s);
        nextClock = defaultClock.getCopy();
        futureClock.makeCopy(currentClock);
        defaultClock.printTime();
        currentClock.printTime();
        nextClock.printTime();
        futureClock.printTime();
}
```

Questions

THANK YOU

Arash Habibi Lashkari

PHD. Candidate of UTM

Kuala Lumpur, Malaysia

Feb, 2010

THE END