CSEN401 – Computer Programming Lab

Topics:

Introduction and Motivation
Recap: Objects and Classes

Prof. Dr. Slim Abdennadher

Course Structure

- **Lectures**
  - Presentation of topics
  - Milestones and assignments
  - Discussion
  - Announcements
- **Labs**
  - Supervised lab work
  - Work in teams
- **Overall weighting for your grade**
  - **30**% for midterm exam
  - **70**% for two projects

Course Policies – Updated

The updated course policies can be found in the policies section of the course on the met website alongside the regulations documents that will be posted

Course Policies – Plagiarism Warning

- Cheating will not be tolerated at any stage of the course
- Plagiarism will be penalized with a deduction of 70% of the course

Team Building and Submission – Second Project

- Teams up to 3 students
- The lab is not a submission of the work of 3 students
- Weekly checks are performed by the teaching assistants to evaluate each student in a team
- Small exams for all groups
- Final Submission of the projects and a competition is planned

Objective of the Course

- In this course, students will learn how to write **object-oriented** code using Java.

Objective of the Course

- In this course, students will learn how to write **object-oriented** code using Java.
- For this purpose, a **real life application** and a **game** will be implemented.

Objective of the Course

- In this course, students will learn how to write **object-oriented** code using Java.
- For this purpose, a **real life application** and a **game** will be implemented.
- **Concepts** will focus on
  - Object-oriented thinking
  - GUI programming (AWT and Swing)

Project: Game

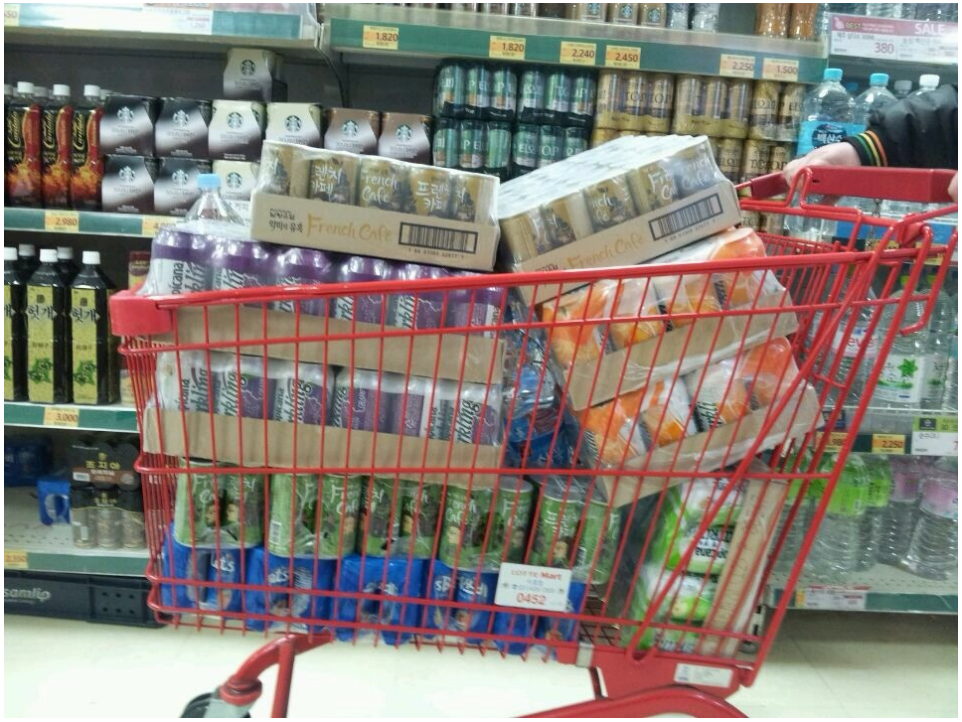
- 3 milestones
- Teams up to 3 students
- Cross-tutorial teams are allowed
- Graded based on successful test cases

Quizzes: Game

- No discussion evaluations in the course
- Quizzes are comprehensive coding tasks
- 1 quiz after each project milestone (3 quizzes)
- Bring your laptop with you
- Graded based on successful test cases
- Final grade of project milestones is dependent on the quizzes grades

```
double ratio= quiz/milestone*100;
double finalGrade=0;
if(ratio>=85)
        finalGrade= milestone;
else if(ratio<85 && ratio>=70)
        finalGrade= (2.0/3)*milestone + (1.0/3)*(quiz/100)*milestone;
else if(ratio<70 && ratio>=60)
        finalGrade= 0.5*milestone + 0.5*(quiz/100)*milestone;
else if (ratio<60 && ratio>=50)
        finalGrade= (1.0/3)*milestone + (2.0/3)*(quiz/100)*milestone;
else
        finalGrade= quiz;
```

Labs: Supermarket Application



- A project targeting Object Oriented thinking
- Simulating operations in a supermarket:
  - Different types of products
  - Buying, selling, ..etc.
- Ungraded project

Object-Oriented Paradigm: Features

Easily remembered as **A-PIE**

- **A**bstraction
- **P**olymorphism
- **I**nheritance
- **E**ncapsulation

Object-Oriented Paradigm: Features

- **Inheritance**: Objects can be defined and created that are specialized types of already-existing objects.
- **Polymorphism**: the ability of objects belonging to different types to respond to method calls to methods of the same name, each one according to the right type-specific behavior.
- **Abstraction**: the ability of a program to ignore the details of an object's (sub)class and work at a more generic level when appropriate.
- **Encapsulation**: Ensures that users of an object cannot change the internal state of the object in unexpected ways.

What is an Object?

- Forget programming for a while.
- Think about things in the world that are **objects**, and things that are not objects.
- It is easier to list things that are objects than to list things that are not objects.
- **Descartes:** Humans view the world in object oriented terms: human brain wants to think about objects, and our thoughts and memories are organized into objects and their relationships.
- One idea of **object-oriented software** is to organize software in a way that matches the thinking style of our object-oriented brains.

Objects and Classes

- **Examples:**
  - **Student** can be described by name, gender, application number ...
  - **Car** can be described by model, make, year, ...
- An **object**: entity that you can manipulate in your programs (by invoking methods)
- A **class**: a template for creating **objects** with **similar features**. It contains variables to represent the attributes and methods to represent the behavior of the objects.
- When a Java application is being run, objects are created and their methods are invoked (are run.)
- A programmer may define a class using Java, or may use **predefined classes** that come in class libraries.
- Creating an object is called **instantiation**.

Objects and Classes – Example I

The `employee` object has

- **attributes** (which are like adjectives)
  - `age`
  - `educationalDegrees`
  - `yearsOfExperience`
  - `jobTitle`
  - `emailAddress`
- **methods** (or actions) the object can perform or undertake while on the job:
  - `wearCompanyT-shirt()`
  - `emailJokesToFriends()`
  - `SurfInternet()`
  - `eatJunkfood()`

Objects and Classes – Example II

The `car` object has

- **attributes**
  - `year`
  - `make`
  - `model`
  - `top-speed`
  - `isRunning`
- behaviors or **methods** (which correspond to actions the driver might take):
  - `start()`
  - `stop()`
  - `isRunning()`
  - `turnLeft()`

Classes and Objects – Car Example

- **Attributes**:
  - `make`: of type `String`
  - `model`: of type `String`
  - `year`: of type `int`
  - `isRunning`: of type `boolean`
- **Methods**:
  - `start()`: the `start` method starts the car by setting its boolean attribute to true; the method does not return anything.
  - `stop()`: the `stop` method stops the car by setting its boolean attribute to false; this method does not return anything.
  - `isRunning()`: the `isRunning` method tells you whether or not the car is running, by returning a boolean value (true if it the car is running).

Instance Variables

Instance Variables are variables to store the state (attributes) of an object.

```
accessSpecifier class ClassName
{   ...
   accessSpecifier VariableType VariableName;
   ...
}
```

- An **access specifier** (usually `private`)
- The **type** of the variable
- The **name** of the variable

```
public class Car
{  // list of 4 attributes
   private String  make ;
   private String  model;
   private int   year;
   private boolean  isRunning;
```

Constructors

The **constructor** is a special type of method

- it initializes the instance variables (set certain values for the instance at creation-time)
- does not specify any return type (not even void)
- must have the **same name as the class** (and Java is case-sensitive)
- can take any number of parameters
- can take any type of parameters

```
accessSpecifier class ClassName
{   ...
   accessSpecifier ClassName(parameterType parameterName ...)
   {
      constructor implementation
   }
   ...
}
```

Constructors – Car Example

```
public class Car
{  // list of 4 attributes
   private String  make ;
   private String  model;
   private int   year;
   private boolean  isRunning;

   //  Constructor, which "initializes" the instance of the class
   public Car(String theMake, String theModel, int theYear)
   {
        make  = theMake;
        model  = theModel;
        year  = theYear;
   }
.....
}
```

Methods – Car Example

```
  //  the first method starts the car
public void start()
  {
      if (isRunning() == false)
           {  isRunning  = true; }
  }

 //  the second method stops the car
public void stop()
{
    if (isRunning())
    isRunning  = false;
}

// the third method returns whether or not the car is running
public boolean isRunning()
{
    return isRunning;
 }
```

Testing a Class

Test class: a class with a `main` method that contains statements to test another class.

Typically carries out the following steps:

- Construct one or more objects of the class that is being tested.
- Invoke one or more methods.
- Print out one or more results.

Class Instantiation: Creating Objects

To create an **instance** (or object) from the class, we use the keyword `new` followed by a call to the constructor.

- **Syntax**:

  ```
  <Class name> <variable name> = new <constructor>;
  ```

- **Result**: The constructor constructs the object and returns a **reference** (variable name) for that newly created object.

- **Example**:

  ```
  public static void main(String[] args)
  {
     Car  myCar  = new Car("Toyota", "Pickup", 1985);
     Car  yourCar = new Car("VW", "Golf", 2004);
  }
  ```

Testing a Class: Accessing Variables and Invoking Methods

```
....
  public static void main ( String[] args )
  {
    Car  myCar  = new Car("Toyota", "Pickup", 1985);


    System.out.println(mycar.year);  // display the year of mycar

    mycar.start();           // invoke the method start() on mycar

    System.out.println(mycar.isRunning); // display the state of mycar
  }
```

Testing a Class: Predefined String Class

```
....
  public static void main ( String[] args )
  {
    String str1;   // str1 is a variable that refers to an object,
                   // but the object does not exist yet.
    int    len;    // len is a primitive variable of type int

    str1 = new String("German University in Cairo");
    // create an object of type String

    len  = str1.length();  // invoke the object's method length()

    System.out.println("The string is " + len + " characters long");
  }
```

Objects and Classes – Example

- All persons are described by a common set of properties or **fields** (**Instance variables**):
  - Name
  - Year of birth
- The **object type** is based on the names and types of its fields.
- The main role of **classes** is to define types of objects

```
public class Person {
   String name;
   int yearOfBirth;
}
```

Constructing Objects – Example

- Each **instance of this class** (object of this type) will have its own copies of the instance variables (field values)
- Create objects of a given class with appropriate field values

```
public class Person {
  String name;
  int yearOfBirth;

  public Person(String n, int yOfB) {
    name = n;
    yearOfBirth = yOfB;
  }
}
```

Making a (virtual) Person

- Declare a variable of appropriate type to hold the `Person` object.
- Call the constructor for `Person` with appropriate arguments.

```
Person pm = new Person("Tony", 1953);
```

Reading an object's data

```
Person pm = new Person("Tony", 1953);
```

pm.name ⇒ "Tony"

pm.yearOfBirth ⇒ 1953

```
Person slim = new Person("Slim", 1967);
```

slim.name ⇒ "Slim"

slim.yearOfBirth ⇒ 1967

Instance Methods (I)

- An **Instance Method** is a subroutine or function designed to work on the current object.
- A method to change the person's name:

```
public void setName(String newName){
    name = newName; }
```

- A method to get the person's name:

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

- A method to display the name and the year of Birth of a person:

```
public void display() {
    System.out.println("Name: " + name);
    System.out.println("Year of Birth: " + yearOfBirth); }
```

Instance Methods (II)

- Instance Methods apply to objects of the class containing the methods

```
    public static void main(String[] args){
     Person pm = new Person("Tony", 1953);
     pm.display();
     pm.setName("Williams");
     pm.display();
}
```

Class Variables

- We want to keep a track of every instance of a Person class.
- If we could have a variable that was **visible** to every instance, we could increment it every time.
- If we declare an instance variable as `static`, it becomes a **class variable**, and can be seen and modified by all instances.
- ```
  public class Person {
    String name;
    int yearOfBirth;
    static int number;

    public Person(String n, int yOfB) {
      name = n;
      yearOfBirth = yOfB;
      number++;
    }
  }
  ```
  ```

Class Methods

- **Instance method** is a method that is invoked from a specific instance of a class that performs some action related to that instance.
- **A class method** is not necessarily associated with a particular object and need not be invoked from an open object.
  - Class methods are declared with the `static` keyword.

```
public  static int totalNumberofPersons() {
   return number;
}
```

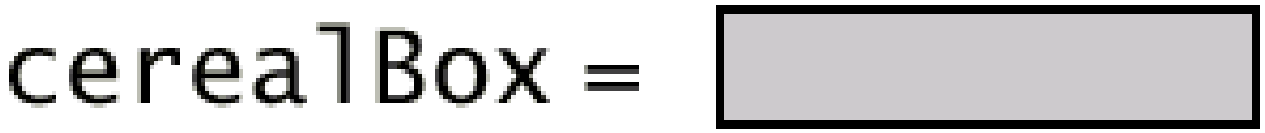
Uninitialized and Initialized Variables

- **Uninitialized Variables**

```
Rectangle cerealBox;
```

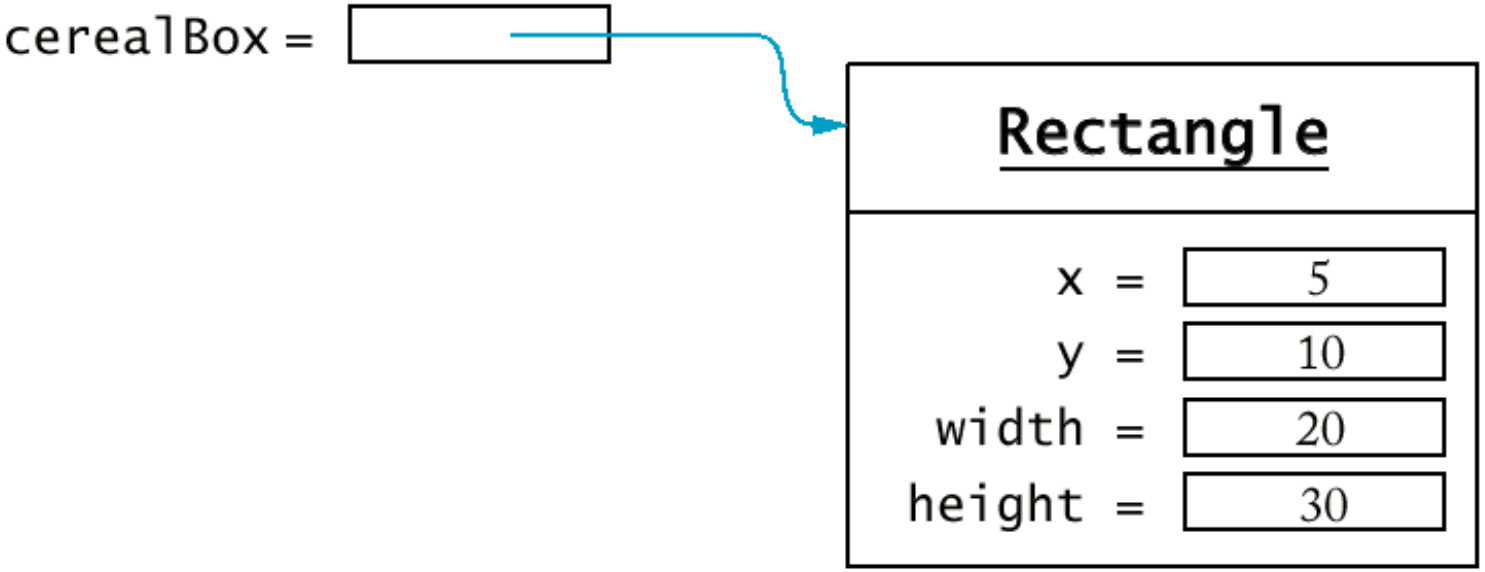
equivalent to

```
Rectangle cerealBox = null;
```



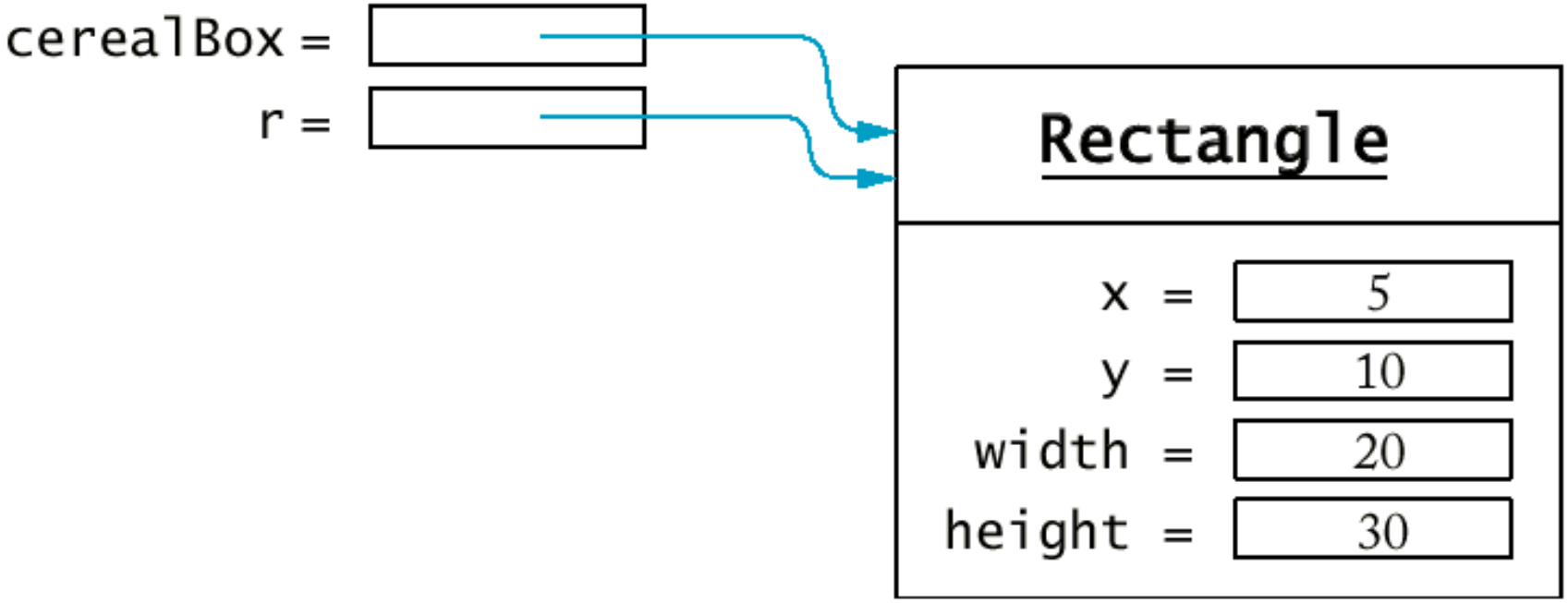
- **Initialized Variables**

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
```



Two Reference Variables Pointing to One Object

```
Rectangle cerealBox = new Rectangle(5, 10, 20, 30);
Rectangle r = cerealBox;
```



Testing Two Reference Variables

```
import java.awt.*;
class Testing
{
  public static void main ( String arg[] )
  {
    Rectangle rectangleA = new Rectangle(5, 10, 20, 30);
    Rectangle rectangleB = new Rectangle(5, 10, 20, 30);

    if (  rectangleA  == rectangleB   )
      System.out.println( "The two variables refer to the same object" )
    else
      System.out.println( "The two variables refer to different objects"

  }
}
```

Result:

```
The two variables refer to different objects
```

Designing and Implementing a Class

- **Step 1:** Find out what you are asked to do with an object of the class.

  Suppose you are asked to implement a `BankAccount` class.

  Operations:

  - deposit money
  - withdraw money
  - get balance

- **Step 2:** Find names for the methods:

```
BankAccount harrysChecking = new BankAccount();
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
harrysChecking.getBalance();
```

- **Step 3:** Determine instance variables

```
private double balance;
```

Designing and Implementing a Class

- **Step 4:** Determine constructors

  Construct a bank account with a given balance:

```
public BankAccount(double initialBalance)
{
   balance = initialBalance;
}
```

- **Step 5:** Implement Methods
- **Step 6:** Test your Class

```
public static void main(String[] args)
   {
      BankAccount harrysChecking = new BankAccount();
      harrysChecking.deposit(2000);
      harrysChecking.withdraw(500);
      System.out.println(harrysChecking.getBalance());
   }
```

BankAccount Class

```
public class BankAccount
{
   private double balance;

   public BankAccount()
   {   balance = 0; }

   public BankAccount(double initialBalance)
   {   balance = initialBalance; }

   public void deposit(double amount)
   {
      double newBalance = balance + amount;
      balance = newBalance;
   }

   public void withdraw(double amount)
   {
```

BankAccount Class

```
        double newBalance = balance - amount;
        balance = newBalance;
    }

    public double getBalance()
    {   return balance; }

    public static void main(String[] args)
    {
        BankAccount harrysChecking = new BankAccount();
        harrysChecking.deposit(2000);
        harrysChecking.withdraw(500);
        System.out.println(harrysChecking.getBalance());
    }
}
```

Designing a simple class

- A point on the plane is given by its coordinates `x`, `y` in a fixed frame of reference

```
class Point {
  // First coordinate.
  double x;
  // Second coordinate.
  double y;
  // Create a new point
  Point(double anX, double aY) {
    x = anX;
    y = aY;
  } }
```

- **Method:** Move the point

```
  void move(double dx, double dy) {
    x += dx;
    y += dy;  }
```

Building on

- A circle is defined by its center (a point) and its radius (a double)

```
class Circle {
  // The center of the circle
  Point center;
  // The radius of the circle
  double radius;

  // Create a  Circle instance
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  } }
```

- **Complex objects**:

```
Point  p = new Point(1,2);
Circle c =  new Circle(p,0.5);
System.out.println(c.center.x);    // 1.0
```

this in instance methods

- within an instance method, `this` refers to the instance being operated on.

```
point move(double dx, double dy) {
  x += dx;
  y += dy;
  return this; }
```

- really means

```
point move(double dx, double dy) {
  this.x += dx;
  this.y += dy;
  return this; }
```

Multiple Constructors

- It is often convenient to construct objects of a type in a variety of ways.
- **Constructor** selected by argument numbers and types

```
class Circle {
  Point center;
  double radius;
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  }
  Circle(double cx, double cy, double aRadius) {
    center = new Point(cx,cy);
    radius = aRadius;
  }
}
```

this in Constructors

- In a **constructor**, `this` can refer to another constructor for the same class

```
class Circle {
  Point center;
  double radius;
  Circle(Point aCenter, double aRadius) {
    center = aCenter;
    radius = aRadius;
  }
  Circle(double cx, double cy, double aRadius) {
    this(new Point(cx,cy),aRadius);
  }
}
```