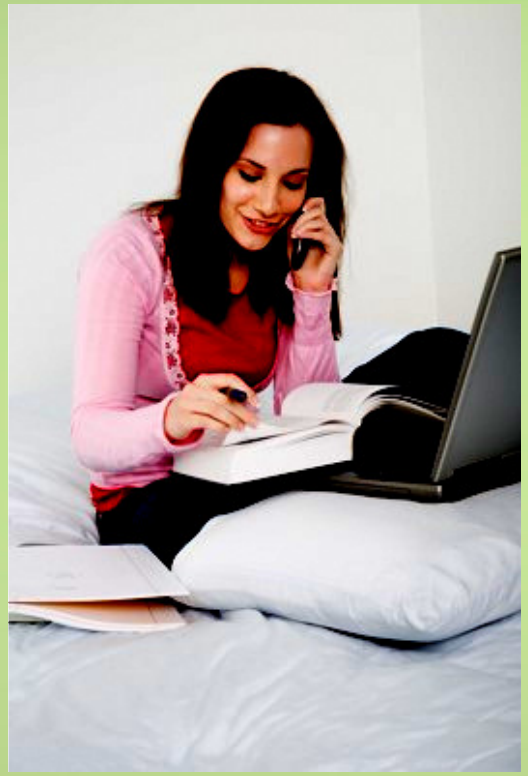
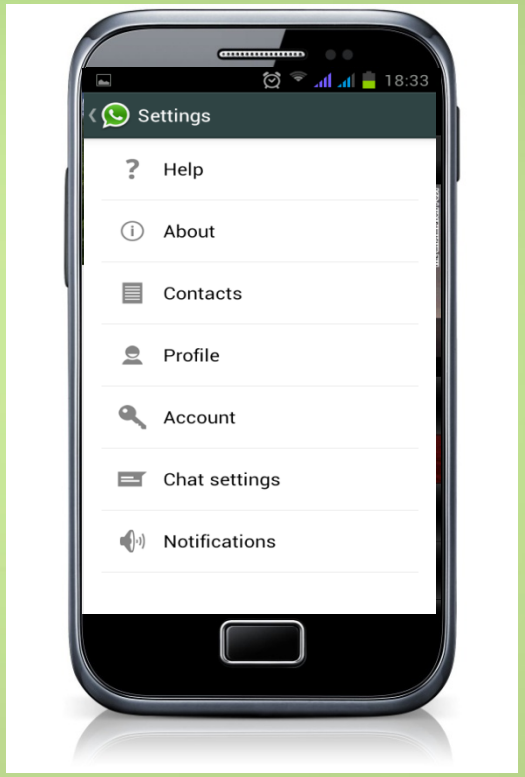
Last Lecture

Elementary Programming

- Primitive data types
- Identifiers and variables
- Assignment statements
- Arithmetic expressions
- Simple I/O

Object Oriented Approach



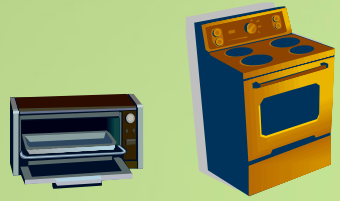


Lecture 3

Object-Oriented Programming

- Classes
- Objects
- Methods
- Scope rules

Object Oriented Programming



- Java is an **object-oriented programming (OOP)** language
  - The concepts of **classes**, **objects and methods** are fundamental
- Goal of this lecture
  - to learn the concepts of classes objects and methods

Cars as Objects

Mary

John

Lily

Tom

TC

Class: Car

Objects: Instances of individual car

- Fields: attributes such as color, no. of passengers, year of made, engineer size
- Methods: behaviors such as move forward, move backward, make turn

Classes

- A class describes a group of objects with common properties and behavior.
- For example:
  - A class of “Car” stands for the general concept of car that moves on wheel and can go from place to place
  - A class of “SmartPhone” are mobile electronic devices that can be used to make phone call, surf the web, play music, send SMS, etc
- The keyword class is used for defining a class
  - e.g. public class Car

    public class SmartPhone
- Designs as a template for creating objects

Objects and Instances

- Once we define a class, we can create instances of objects in that class
- Objects are often referred to as instances
- For example:
  - In a “Car” class, there could be difference instances (objects) which belong to TC, Mary, John, etc.
  - In a “SmartPhone” class, a difference instance can be created for each student in this classroom.
- Instances of objects are created using constructor and the keyword new.

Properties and Methods

- Properties (fields)
  - Values that are owned by an object
  - Examples: owner of a car, color of a car
    brand and model of a smartphone
- Methods
  - Actions that can be performed
  - Examples:
    - A car can move forward, move backward and make turns.
    - A smartphone can make phone call, surf the web, play music and send SMS.

An Example: Car


```
import comp102x.IO;
/**
 *  A class of Car objects that can move forward, backward and turn
 */
public class Car {
```

Instance variables

```
    private int odometer = 0;                  // An odometer reading initialized to 0
    private String owner = "NoName";           // Name of owner
```

Constructor declarations

```
    /**
     *  Default constructor for a Car object
     */
    public Car ( ) { }
    /**
     *  Constructor for a Car object with a new owner's name
     *  @param   name              name of owner
     */
    public Car (String name) {
        owner = name;
    }
```

An Example

Method definitions

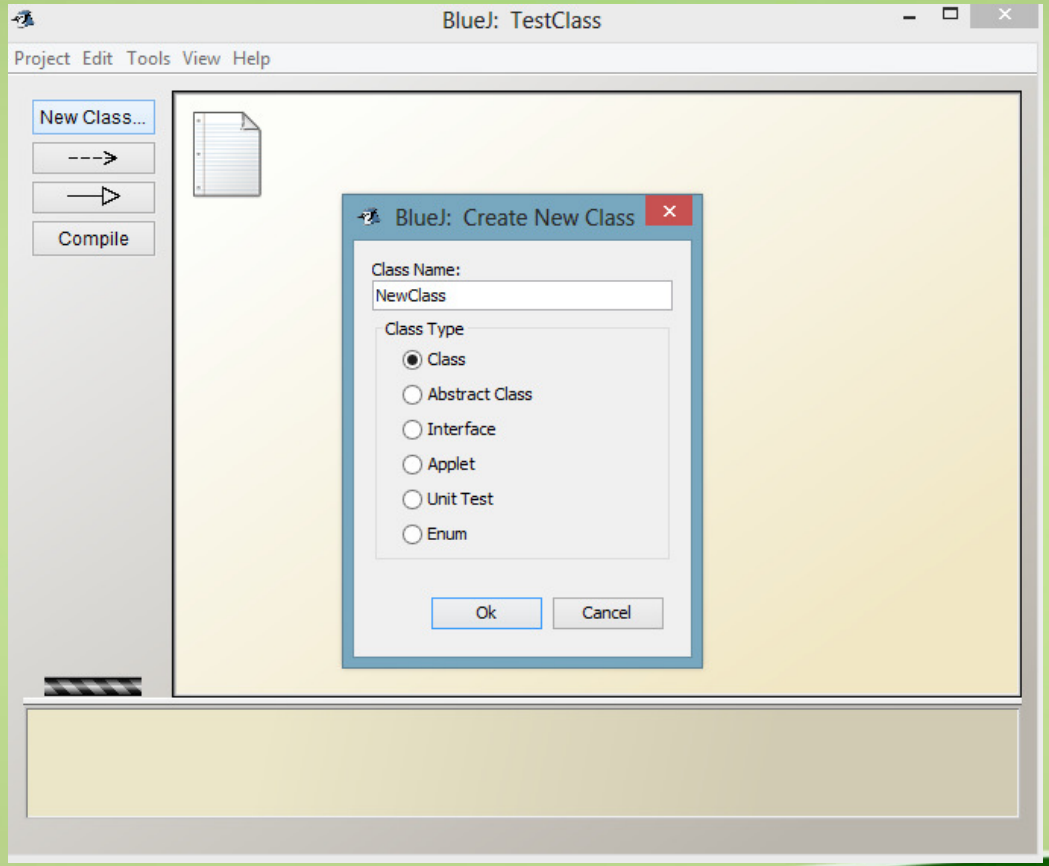
```
/**
 *   moveCar moves a car forward or backward by dist units
 *   @param   dist     moving distance
 */
public void moveCar (int dist) {
    odometer = odometer + dist;
    IO.outputln(owner + "'s car has moved " + dist + " units.");
}
```

An Example

Method definitions

```
/**
 *  moveCar moves a car forward or backward by dist units
 *  @param   dist    moving distance
 */
public void moveCar (int dist) {
    odometer = odometer + Math.abs(dist);
    IO.outputln(owner + "'s car has moved " + dist + " units.");
}
/**
 *  turnCar turns a car by a given degree
 *  @param   angle    turn angle in degrees
 */
public void turnCar(double angle) {
    IO.outputln(owner + "'s car has turned " + angle + " degrees.");
}
/**
 *  getOdometer gets the odometer reading of a car
 */
public int getOdometer( ) {
    return odometer;
}
}
```

General Structure of a Class declaration



NewClass - TestClass

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
/**
 * Write a description of class NewClass here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

General Structure of a Class declaration

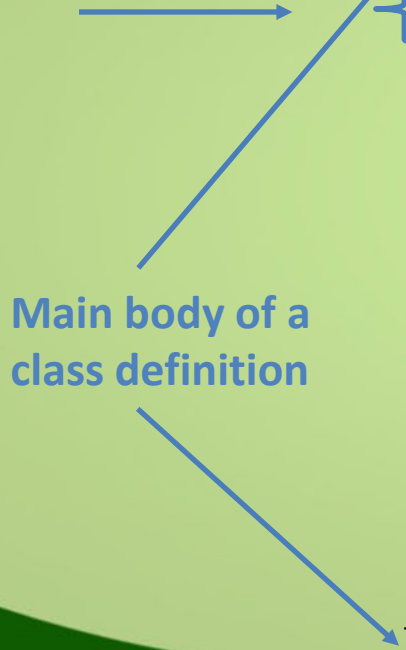
Example:
public class Car

```
/**
 * Write a description of class NewClass here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

Access Identifiers

- Java defines four levels of access identifiers
  - Only public and private will be covered in this course
- Public identifier
  - The class, data or method is visible to any class in any package
- Private identifier
  - The data or methods can only be accessed within the same class

General Structure of a Class declaration




```
/**
 * Write a description of class NewClass here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

General Structure of a Class declaration

Constructors

```
/**
 * Write a description of class NewClass here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

Constructor

- Constructors are declared by using the name of the class as identifier, e.g. NewClass , Car
  - It is used to initialize an object's properties
  - It is invoked once and only once at the time of object creation using the new operator
  - It has no return type
  - Syntax of constructor:
    - public nameOfClass (*parameters*)
    - *Parameters* are optional: public nameOfClass ( )

Constructor

- Examples:

Body of a constructor

```
public NewClass()
 {
       x = 0;          // initialize instance variable
 }
```

```
public Car ( ) { }                  // Constructor with no parameter

public Car (String name)            // Constructor with one parameter
 {
       owner = name;
 }
```

General Structure of a Class declaration

```
/**
 * Write a description of class NewClass here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

Methods

Components of a method

- Each method has 5 major components
  - The access identifier (i.e. public, private)
  - The return type (e.g. void, int, boolean…)
  - The method name
  - The parameter list (optional)
  - The method body

General Structure of a Class declaration

```
/**
 * Write a description of class NewClass here.
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

sampleMethod

An Example: Car

```
import comp102x.IO;
/**
 *  A class of Car objects that can move forward, backward and turn
 */
public class Car  {
    private int odometer = 0;                  // An odometer reading initialized to 0
    private String owner = "NoName";           // Name of owner

    /**
     *  Default constructor for a Car object
     */
    public Car ( ) { }
    /**
     *  Constructor for a Car object with a new owner's name
     *  @param   name                name of owner
     */
    public Car (String name) {
        owner = name;
    }
```

Instance variables

Constructor declarations

An Example

Method definitions

```
    /**
     *   moveCar moves a car forward or backward by dist units
     *   @param   dist     moving distance
     */
    public void moveCar (int dist)  {
        odometer = odometer + Math.abs(dist);
        IO.outputln(owner + "'s car has moved " + dist + " units.");
    }
    /**
     *   turnCar turns a car by a given degree
     *   @param   angle     turn angle in degrees
     */
    public void turnCar(double angle)  {
        IO.outputln(owner + "'s car has turned " + angle + " degrees.");
    }
    /**
     *   getOdometer gets the odometer reading of a car
     */
    public int getOdometer( )  {
        return odometer;
    }
}
```

Documenting a Program

- Program documentation is provided in the form of comments. Typically, comments are used to describe the following:
  - What a program does and requires
  - What a variable represents
  - What care should be taken when using a variable or a method
  - A brief description of the logic flow of the program
  - Some information about the author and the code itself

- There are three general ways to put comments in your code, so that the computer will not treat it as part of the programming statements

Method 1: // Line comment

- A line comment is one line of sentence preceded by two forward slashes (//)
  - A line comment is usually placed on top or on the right-hand side of a programming statement:
- **Examples:**

```
String owner = "NoName";    // Name of owner


// Method to move the car forward
public void moveForward( )
{ IO.outputln(owner + "'s car is moving forward."); }
```

Method 2: /* Paragraph comment */

- A paragraph comment is enclosed between /* and */ in one or multiple lines
  - It is usually used when a detailed description is required to explain a major section of codes
- Example:

```
/*
 * Compute final grade as the weighted some of exam scores, lab scores
 * and homework scores. Note that all scores should be within the range
 * of 0 and 100
*/
        examScore = examScore * (examWeight / 100.0);
        labScore = labScore * (labWeight / 100.0);
        hwScore = hwScore * (hwWeight / 100.0);
        finalGrade = examScore + labScore + hwScore;
```

Method 3: /** JavaDoc comment */

Javadoc: Useful tool for generating documentation from Java programs

- Begins with /** and ends with */
- @param for describing parameters
- @return for describing the return value of a method
- Generates document in HTML format that can be displayed as webpages

NewClass - TestClass

Class Edit Tools Options

Compile | Undo | Cut | Copy | Paste | Find... | Close | Source Code

```
/**
 * Write a description of class NewClass here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NewClass
{
    // instance variables - replace the example below with your own
    private int x;

    /**
     * Constructor for objects of class NewClass
     */
    public NewClass()
    {
        // initialise instance variables
        x = 0;
    }

    /**
     * An example of a method - replace this comment with your own
     *
     * @param  y   a sample parameter for a method
     * @return     the sum of x and y
     */
    public int sampleMethod(int y)
    {
        // put your code here
        return x + y;
    }
}
```

All Classes

NewClass

Constructor Detail

NewClass

```
public NewClass()
```

Constructor for objects of class NewClass

Method Detail

sampleMethod

```
public int sampleMethod(int y)
```

An example of a method - replace this comment with your own

Parameters:

`y` - a sample parameter for a method

Returns:

the sum of x and y

NewClass - TestClass

Class Edit Tools Options

Compile Undo Cut Copy Paste Find... Close Documentation

Class NewClass

java.lang.Object
  └ NewClass

public class NewClassextends Object

Write a description of class NewClass here.

Version:
(a version number or a date)
Author:
(your name)

Constructor Summary

NewClass() Constructor for objects of class NewClass

Method Summary

sqrt

```
public static double sqrt(double a)
```

Returns the correctly rounded positive square root of a `double` value. Special cases:

- If the argument is NaN or less than zero, then the result is NaN.
- If the argument is positive infinity, then the result is positive infinity.
- If the argument is positive zero or negative zero, then the result is the same as the argument.

Otherwise, the result is the `double` value closest to the true mathematical square root of the argument value.

Parameters:

`a` - a value.

Returns:

the positive square root of `a`. If the argument is NaN or less than zero, the result is NaN.

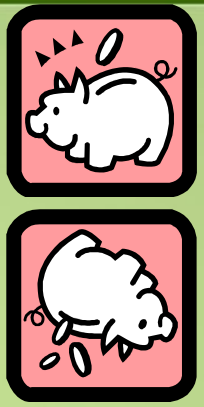
cbrt

```
public static double cbrt(double a)
```

Returns the cube root of a `double` value. For positive finite `x`, `cbrt(-x) == -cbrt(x)`; that is, the cube root of a negative value is the negative of the cube root of that value's magnitude. Special cases:

- If the argument is NaN, then the result is NaN.

An Example: Bank Account



```
import comp102x.IO;
/**
 * A bank account has a balance and an owner who can make
 * deposits to and withdrawals from the account.
 */
public class BankAccount {
```

Instance variables

```
   private double balance = 0.0;      // Initial balance is set to zero
   private String owner = "NoName";     // Name of owner
```

Constructors

```
   /**
    *   Default constructor for a bank account with zero balance
    */
   public BankAccount ( ) { }
   /**
    *   Construct a balance account with a given initial balance and owner's name
    *   @param   initialBalance     the initial balance
    *   @param   name               name of owner
    */
   public BankAccount (double initialBalance, String name)  {
      balance = initialBalance;
      owner = name;
   }
```

An Example: Bank Account

Method definitions

```
/**
 *  Method for depositing money to the bank account
 *  @param  dAmount    the amount to be deposited
 */
public void deposit(double dAmount)  {
    balance = balance + dAmount;
}
/**
 *  Method for withdrawing money from the bank account
 *  @param  wAmount    the amount to be withdrawn
 */
public void withdraw(double wAmount) {
    balance = balance - wAmount;
}
/**
 *  Method for getting the current balance of the bank account
 *  @return  the current balance
 */
public double getBalance() {
    return balance;
}
```

An Example: Bank Account

```
/**
 *  Method for depositing money to the bank account
 *  @param  dAmount    the amount to be deposited
 */
public void deposit(double dAmount)  {
    balance = balance + dAmount;
}
/**
 *  Method for withdrawing money from the bank account
 *  @param  wAmount    the amount to be withdrawn
 */
public void withdraw(double wAmount) {
    balance = balance - wAmount;
}
/**
 *  Method for getting the current balance of the bank account
 *  @return  the current balance
 */
public double getBalance() {
    return balance;
}
```

Mutator or setter methods

Accessor / getter method

香港科技大學
THE HONG KONG UNIVERSITY OF SCIENCE AND TECHNOLOGY

An Example

```
/**
 *   Main method for testing the bank account
 */
public static void main (String[ ] args) {
    BankAccount testAccount;
    testAccount = new BankAccount( );




  }
}
```

An Example

```
/**
 *   Main method for testing the bank account
 */
  public static void main (String[ ] args) {
     BankAccount testAccount = new BankAccount( );
     testAccount.deposit(100);
     testAccount.withdraw(50);
     IO.outputln (testAccount.owner + "'s account has a balance of $"
              + testAccount.balance);

     BankAccount myAccount = new BankAccount(100, "TC");
     myAccount.deposit(100);
     myAccount.withdraw(50);
     IO.outputln (myAccount.owner + "'s account has a balance of $"
              + myAccount.balance);
  }
}
```

Class ColorImage

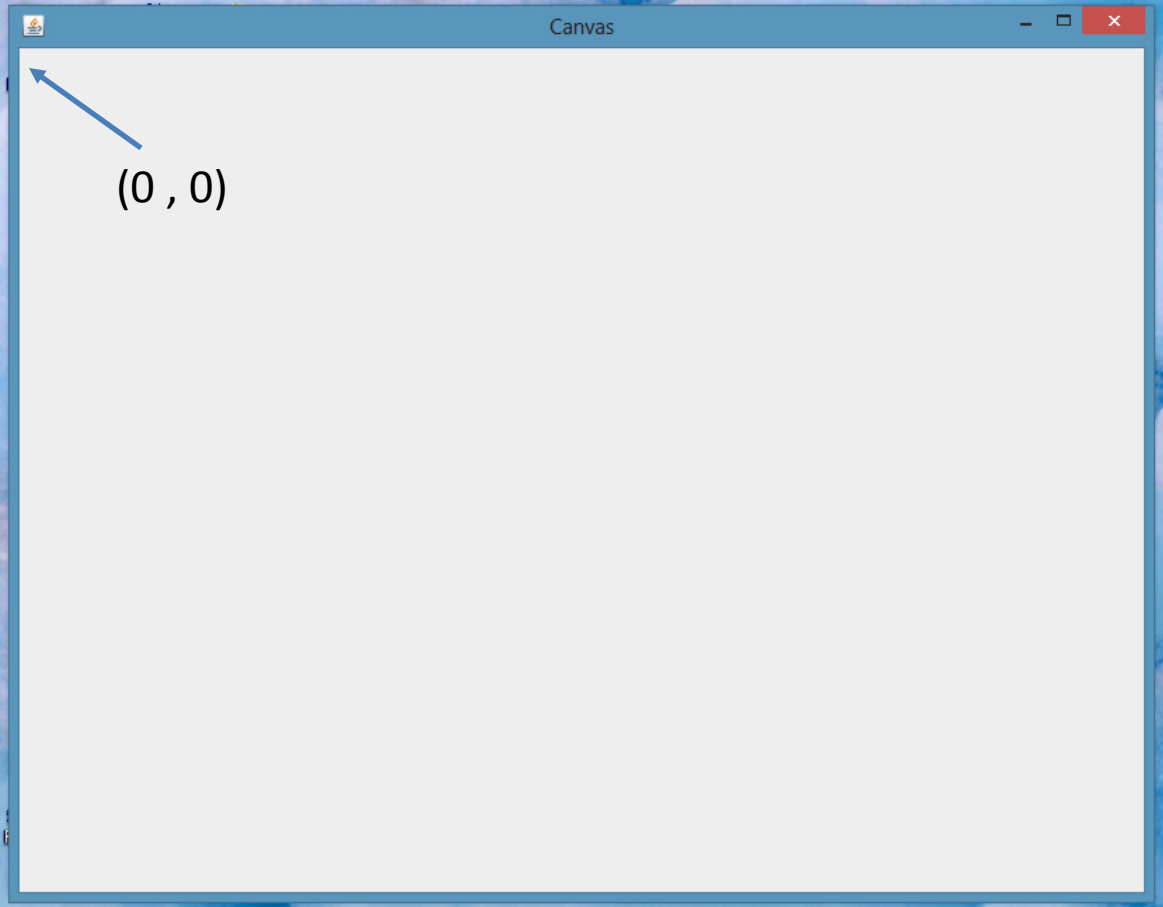
java.lang.Object
    CanvasObject
        ColorImage

```
public class ColorImage
extends CanvasObject
```

Constructor Summary

Constructors

Constructor and Description
ColorImage() Construct a color image object by loading an image from the file system
ColorImage(int width, int height) Construct a blank white color image object with a specify width and height
ColorImage(java.lang.String filename) Construct a color image object by using a filename


Canvas
(0 , 0)

Example: ColorImage Class

Instance variable

Constructors

```
/**
 * A simple demo on ColorImage.
 */
public class ColorImageDemo
{
   private Canvas canvas = new Canvas();      // Create a canvas for ColorImage
   private ColorImage image1 = new ColorImage("Car1.png");  // Default image

   // Define two constructors for ColorImageDemo
   public ColorImageDemo() {
      canvas.add(image1, 0, 0);            // Display ColorImage at (0,0) position
   }

   public ColorImageDemo(int xPos, int yPos) {
      image1 = new ColorImage( );         // Create a new ColorImage from user file
      canvas.add(image1, xPos, yPos);     // Display ColorImage at (xPos,yPos) position
   }
```

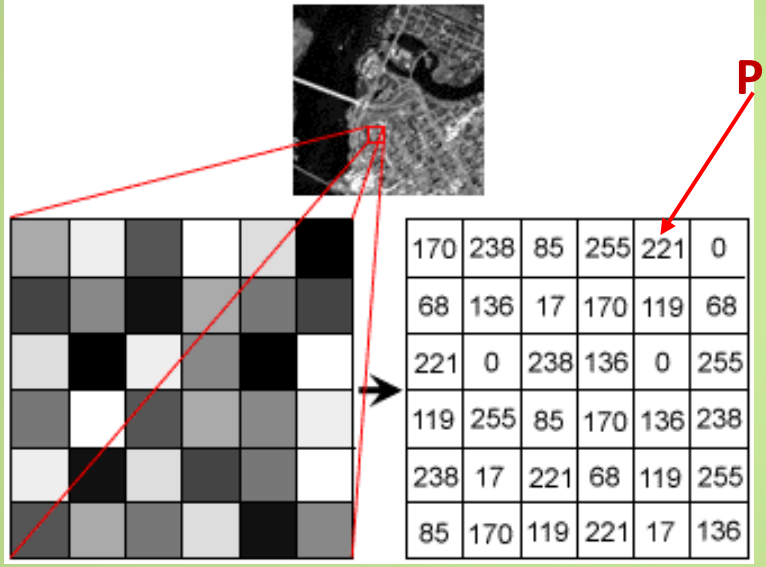
Example: ColorImage Class

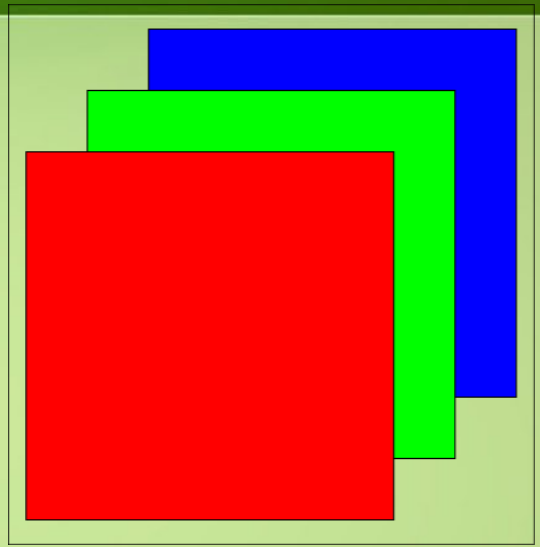
Method definitions

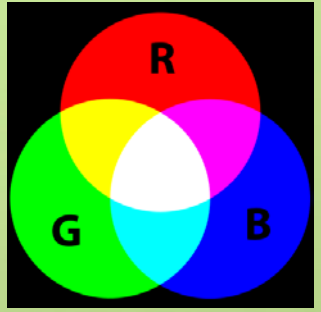
```
// Rotate the image clockwise by degrees
public void setRotateDemo(int degrees) {
   image1.setRotation(degrees);
}
// Get the degrees in clockwise rotation of the image
public int getRotateDemo( ) {
   return image1.getRotation();
}
// Scale the image by scaleFactor
public void scaleDemo(double scaleFactor) {
   image1.setScale(scaleFactor);
}
// Move the image to position (x,y) on the canvas
public void translateDemo(int x, int y) {
   image1.setX(x);
   image1.setY(y);
}
}
```

How to change the method so that the translation will start from the current position instead of the origin?

ColorImage


Pixel
170 238 85 255 221 0
68 136 17 170 119 68
221 0 238 136 0 255
119 255 85 170 136 238
238 17 221 68 119 255
85 170 119 221 17 136



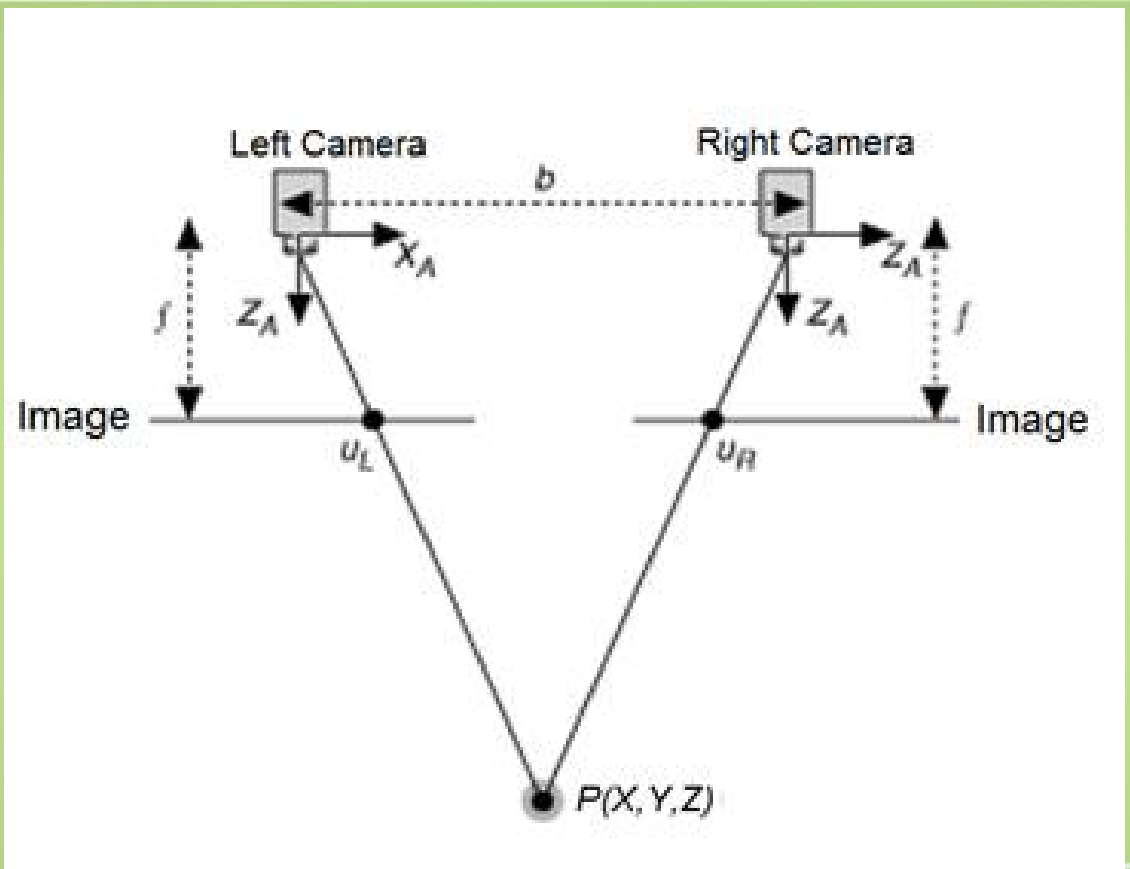

R
G
B

ColorImage

Methods

Modifier and Type	Method and Description
`ColorImage`	**`add`**`(ColorImage operand)` Add another ColorImage object to this ColorImage object
`static ColorImage`	**`add`**`(ColorImage image1, ColorImage image2)` Add two ColorImage objects together
`void`	**`add`**`(int value)` Add a value to all the RGB channels of this ColorImage object
`double`	**`averageDifference`**`(ColorImage operand)` Get the average difference between this ColorImage object and the other ColorImage object
`void`	**`convolve`**`(float[][] kernel)` Convolve the ColorImage object with a specified kernel
`ColorImage`	**`createCyanImage`**`()` Create a copy of this ColorImage object with the red channel removed
`ColorImage`	**`createRedImage`**`()` Create a copy of this ColorImage object with the green and blue channel removed
`void`	**`decreaseBlue`**`(int value)` Decrease the value of blue channel for all pixels
`void`	**`decreaseGreen`**`(int value)` Decrease the value of green channel for all pixels
`void`	**`decreaseRed`**`(int value)` Decrease the value of red channel for all pixels

Stereo / 3D Images



The Car Example

```
//   A class of Car objects that can move forward, backward and turn
public class Car2
{

   private String owner = "NoName";
   private ColorImage carImage = new ColorImage("Car1.png");
   private double gasMileage = 10.0;  // Liters for every 100km
   private double gasInTank = 10.0;
```

Instance variables

Class Car2
Owner [String] : "NoName"
carImage [ColorImage] : 
gasMileage [double] : 10.0
gasInTank [double] : 10.0

The Car Example

Constructors

```
public Car2 ( ) { }

public Car2 (String nameOfOwner)
{
   owner = nameOfOwner;
   carImage = new ColorImage( );
}

public Car2 (String nameOfOwner, double newGasMileage)
{
   owner = nameOfOwner;
   carImage = new ColorImage( );
   gasMileage = newGasMileage;
}
```

The Car Example

```
public void moveForward(int dist) {
    IO.outputln(owner + "'s car is moving forward.");
}
```

Methods

The Car Example

Methods

```
public void moveForward(int dist) {
    // Change the X position of car from current X postion plus dist
    carImage.setX(carImage.getX() + dist);
    // Update the amount of gas in tank
    double gasUsed = dist / 100.0 * gasMileage;
    gasInTank = gasInTank – gasUsed;
    IO.outputln("Amount of gas used: " + gasUsed + ", gas remained: " + gasInTank);
}
 public void makeTurn(int angle) {
    // Change the orientation of car from current orientation plus angle
    carImage.setRotation(carImage.getRotation() + angle);
 }
 // addGas adds an amount of gas equal to gasToAdd to gasInTank
 public void addGas( ) {
     gasInTank = gasInTank + gasUsed;
 }
}
```

The Car Example

Methods

```
public void moveForward(int dist) {
    // Change the X position of car from current X postion plus dist
    carImage.setX(carImage.getX() + dist);
    // Update the amount of gas in tank
    double gasUsed = dist / 100.0 * gasMileage;
    gasInTank = gasInTank – gasUsed;
    IO.outputln("Amount of gas used: " + gasUsed + ", gas remained: " + gasInTank);
}
 public void makeTurn(int angle) {
    // Change the orientation of car from current orientation plus angle
    carImage.setRotation(carImage.getRotation() + angle);
 }
 // addGas adds an amount of gas equal to gasToAdd to gasInTank
  public void addGas(double gasToAdd) {
     gasInTank = gasInTank + gasToAdd;
 }
}
```