Chapter 2

An Overview

Declaration

- These slides are made for UIT, BU students only. I am not holding any copy write of it as I had collected these study materials from different books and websites etc. I have not mentioned those to avoid complexity.

Syllabus

- **Features of object-oriented (OO) programming:** Encapsulation, object identity, polymorphism – but not inheritance.
- **Inheritance in OO design**

Two Paradigms

- All computer programs consist of two elements: code and data.
- A program can be conceptually organized around its code or around its data.
- Some program is written around "what is happening" and others are written around "what is being affected"
- The first way is called process-oriented model – characterizes a program as a series of linear steps – **code acting on data**
- Second approach is object oriented programming (OOP) – to manage increasing complexity.
- It organizes a program around its data (that is, objects) and a set of well defined interfaces to that data.
- **Data controlling access to code**.

Two Paradigms

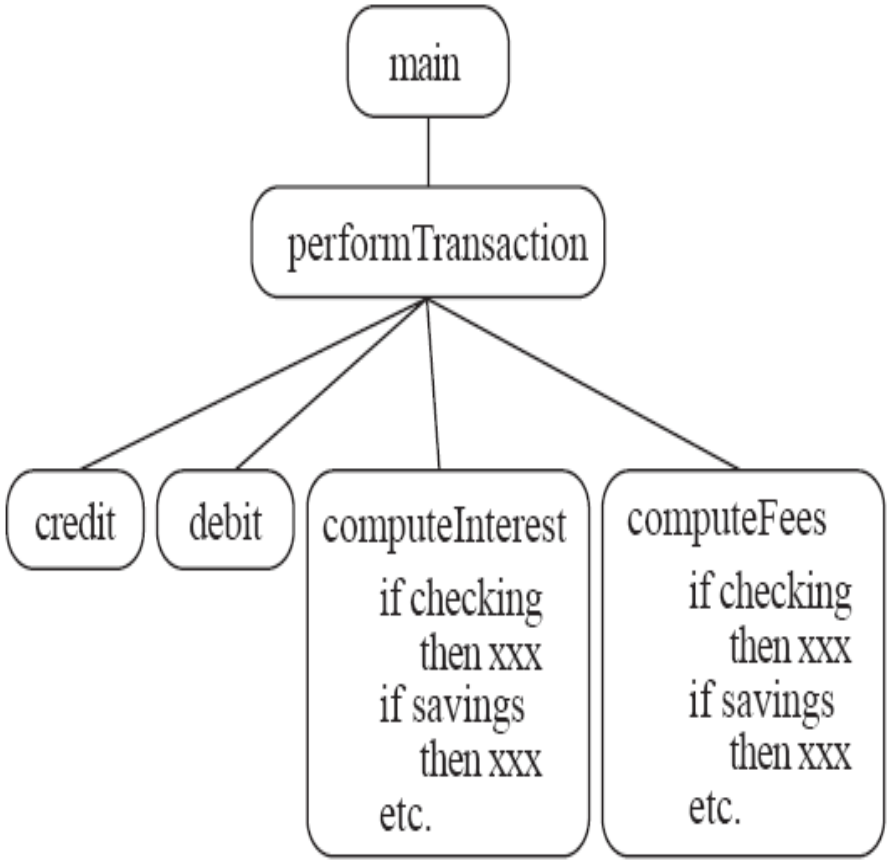
Two complementary ways to view software construction:

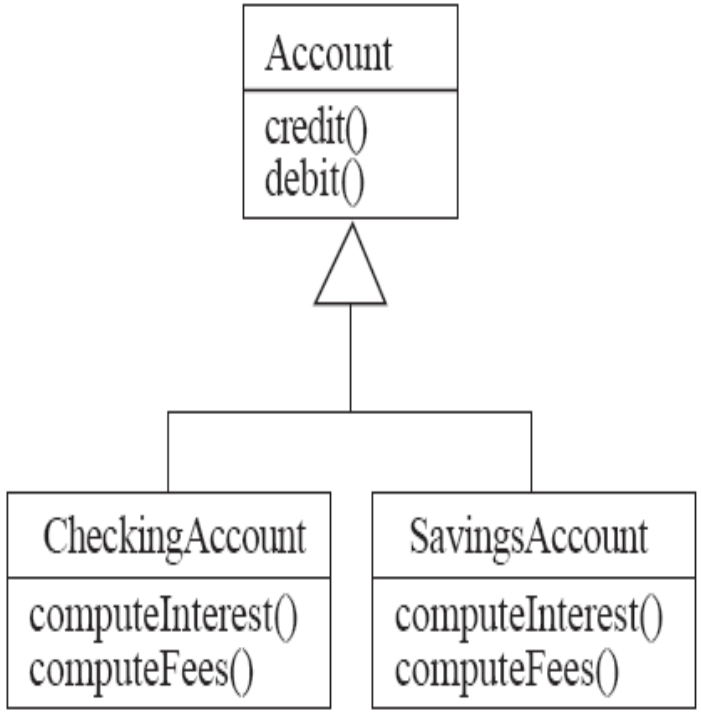
1. **Procedural approach**: focusing primarily on function.
2. **Object Oriented approach**: focusing primarily on data.

Two Paradigms

- In procedural approach
  - If we want to add two integer number then we have to write a program
  - If we want to add two decimal number then we have to write another program
  - Code will decide data that program will handle
- In OOP
  - We can write two methods in a class
  - One method will take two integer numbers as input and return the summation after adding numbers
  - Another method with same name will take two decimal (float) numbers as input and return the summation after adding numbers
  - Here data will decide which function will be call

Two Paradigms

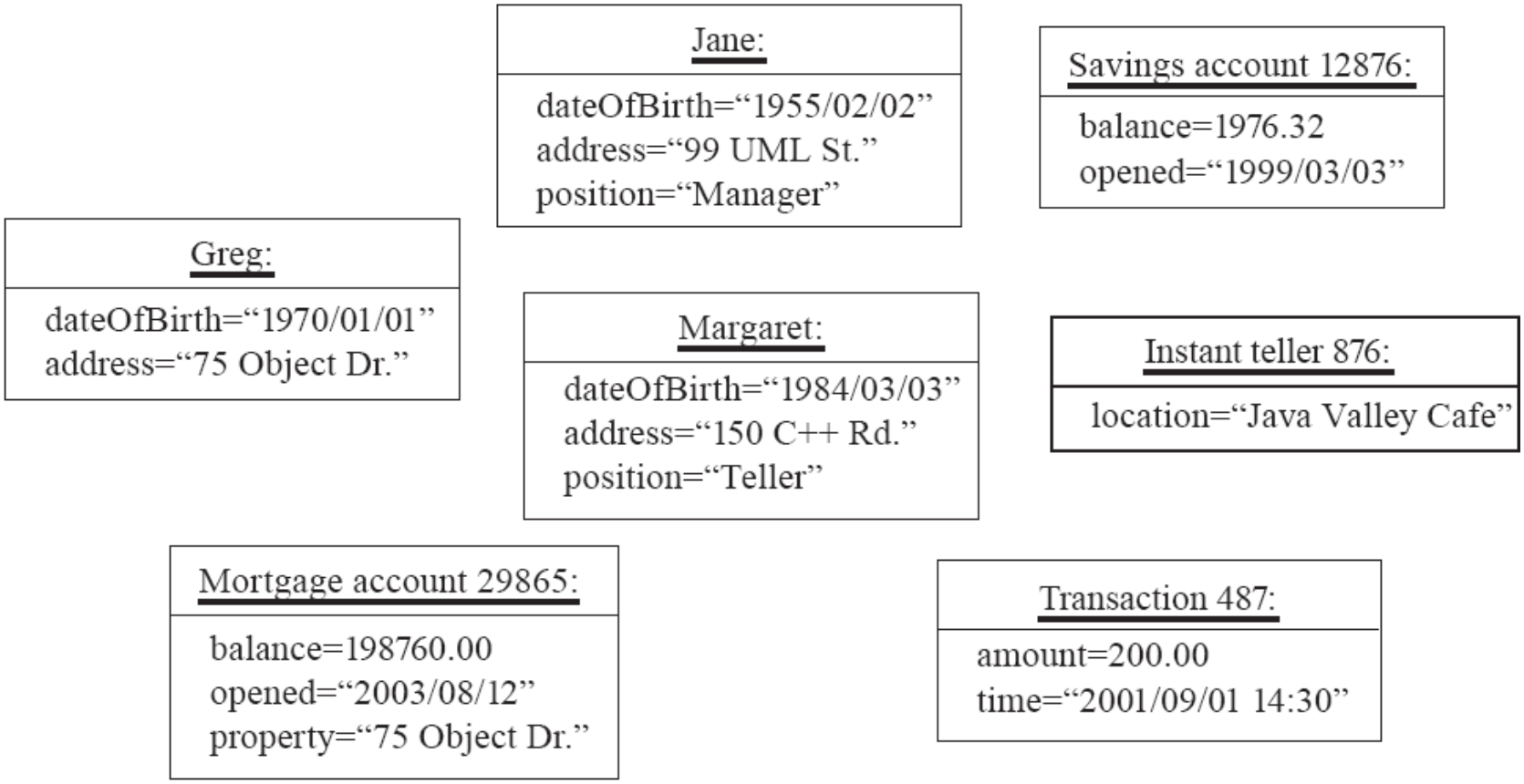




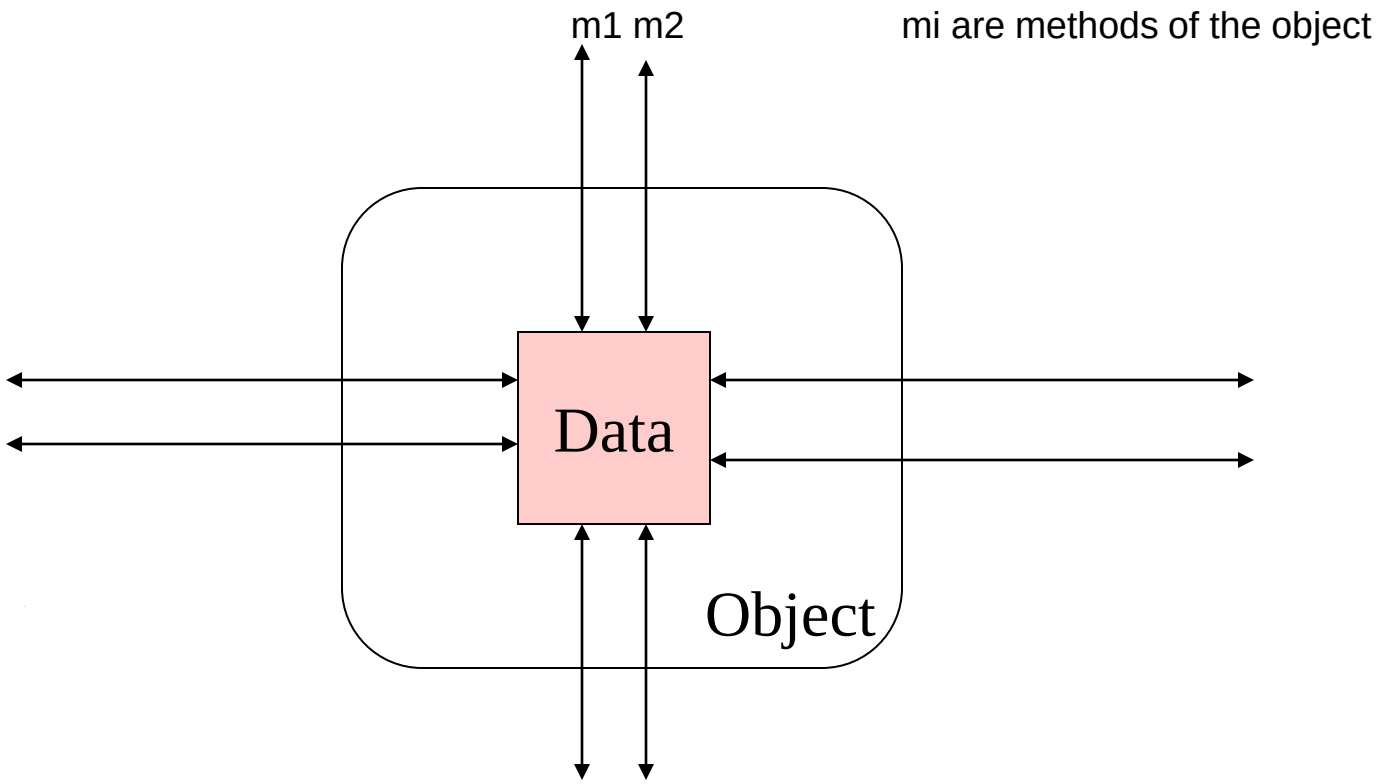
Objects

- An object, in object-oriented programming (OOP), is an abstract data type created by a developer. It can include multiple properties and methods and may even contain other objects.
- In most programming languages, objects are defined as classes.

Objects


Jane:
dateOfBirth="1955/02/02"
address="99 UML St."
position="Manager"
Savings account 12876:
balance=1976.32
opened="1999/03/03"
Greg:
dateOfBirth="1970/01/01"
address="75 Object Dr."
Margaret:
dateOfBirth="1984/03/03"
address="150 C++ Rd."
position="Teller"
Instant teller 876:
location="Java Valley Cafe"
Mortgage account 29865:
balance=198760.00
opened="2003/08/12"
property="75 Object Dr."
Transaction 487:
amount=200.00
time="2001/09/01 14:30"

Object-Oriented Concepts



Model of an object

Abstruction

- An essential element of OOP.
- Human manage complexity through abstraction.
- For example: car
- To manage abstraction – hierarchical classification
- The data from traditional process oriented program can be transformed by abstraction into its compound objects.
- A sequence of process steps can become a collection of messages between these objects.
- Each of these objects describes its own unique behavior.
- These are concrete entities that respond to messages telling them to do something.

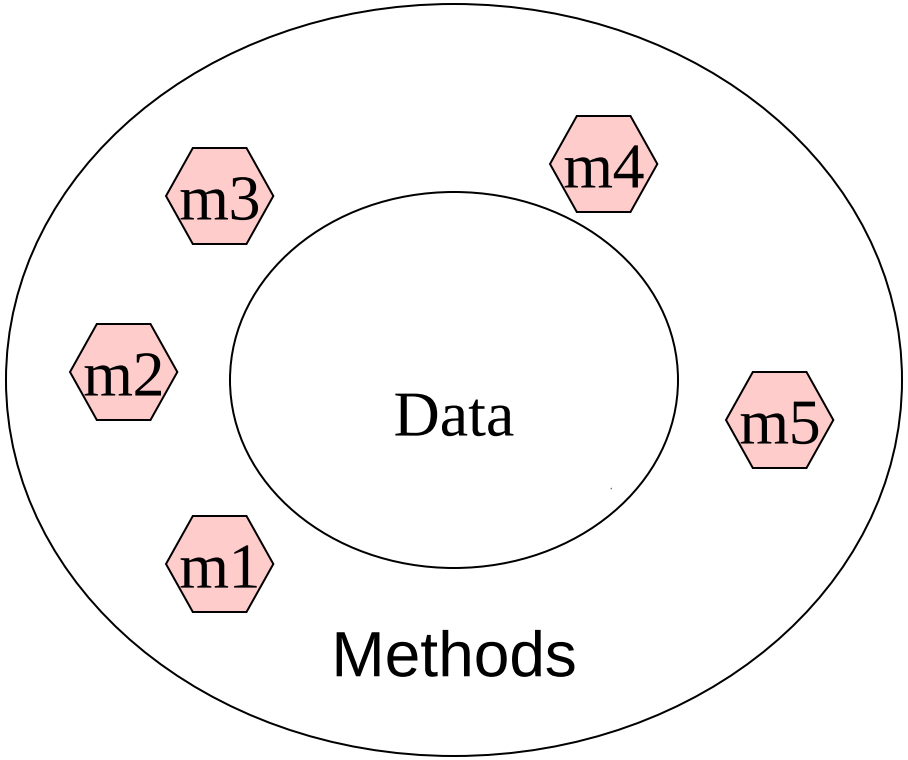
OOP principles

- Three OOP principles are
  - Encapsulation
  - Inheritance
  - Polymorphism

Encapsulation

- A mechanism which hides code and data
- It keeps both safe from outside interference and misuse.
- Access of code and data is tightly controlled through a well defined interface.
- In java basis of encapsulation is class.
- Class defines the structure and behavior (code and data) that will be shared by a set of objects.
- Objects are instances of class.
- Data – member variables – instance variables
- Code – member methods – methods
- Public interface – external user of the class need to know.
- Private interface – access by code that is member of the class.

Encapsulation



Concept of encapsulation

Inheritance

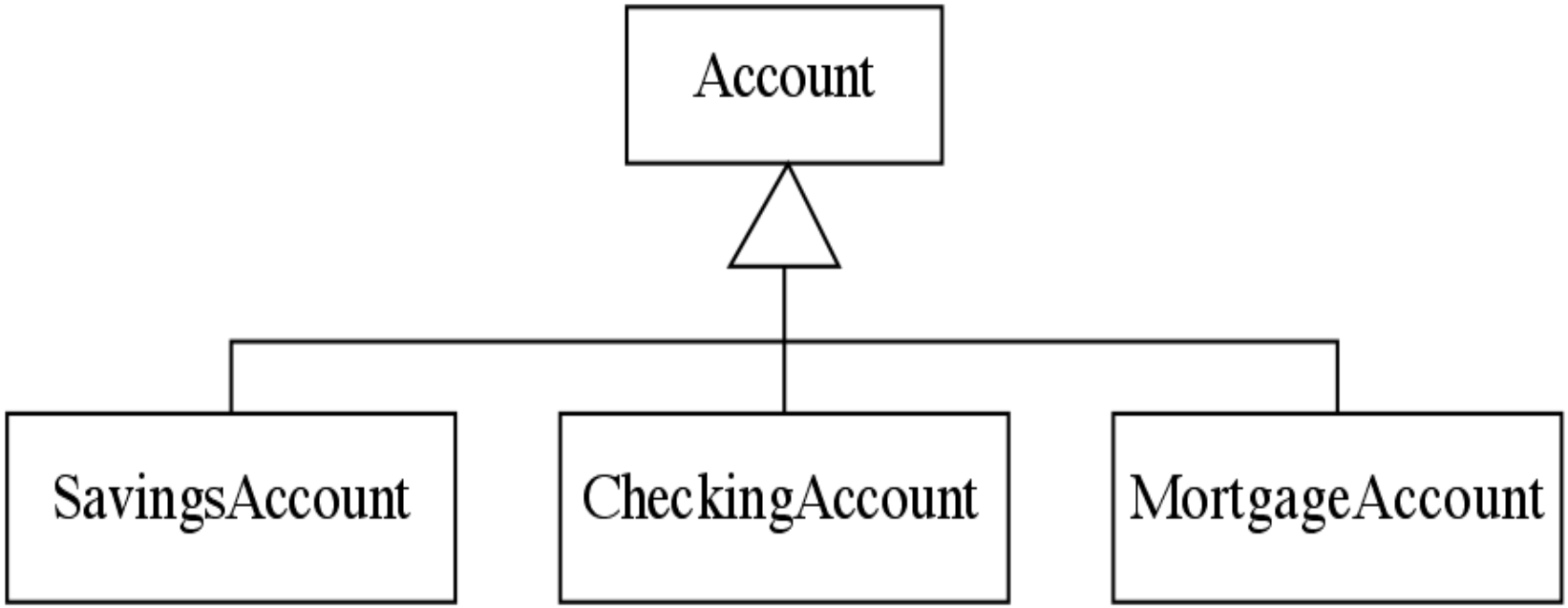
- By this objects acquires the property of other object.
- Supports the concept of hierarchical classification.
- Allows to define a new class (derived class) by extending or modifying existing class (base class).

– Represents generalization-specialization relationship.

– Allows redefinition of the existing methods (method overriding).

– Allows reuse of code

Inheritance



Inheritance

- The **implicit** possession by all subclasses of features defined in its superclasses

Polymorphism

- Ability to take more than one form.
- An operation may exhibit different behavior in different instances.
- The behavior depends upon the types of data in the operation.
- Example – addition of integer number and string

Advantages of Object-Oriented Development

- Code and design reuse
- Increased productivity
- Ease of testing (?) and maintenance
- Better understandability
- Elegant design: loosely coupled, highly cohesive objects:

– Very useful for solving large problems.

Advantages of Object-oriented Development

- Initially incurs higher costs

– After completion of some projects reduction in cost become possible

- Using well-established OO methodology and environment:

– Projects can be managed with 20%-50% of traditional cost of development.

Installation and testing Java

- This article applies to: **Platform(s):** Windows 2008 Server, Windows 7, Windows 8, Windows XP, Windows Server 2012, Windows Vista, Windows 10
- **Java version(s):** 7.0, 8.0
- It is recommended, before you proceed with online installation you may want to disable your Internet firewall. In some cases the default firewall settings are set to reject all automatic or online installations such as the Java online installation. If the firewall is not configured appropriately it may stall the download/install operation of Java under certain conditions. Refer to your specific Internet firewall manual for instructions on how to disable your Internet Firewall.

Installation and testing Java

- Go to the Manual download page
- Click on **Windows Online**
- The File Download dialog box appears prompting you to run or save the download file
  - To run the installer, click **Run**.
  - To save the file for later installation, click **Save**.
    Choose the folder location and save the file to your local system.
    Tip: Save the file to a known location on your computer, for example, to your desktop.
    Double-click on the saved file to start the installation process.
- The installation process starts. Click the **Install** button to accept the license terms and to continue with the installation.

Installation and testing Java

- Oracle has partnered with companies that offer various products. The installer may present you with option to install these programs when you install Java. After ensuring that the desired programs are selected, click the **Next** button to continue the installation.
- A few brief dialogs confirm the last steps of the installation process; click **Close** on the last dialog. This will complete Java installation process.

A Simple Program: Printing a Line of Text

```
1 // Welcome1.java
2 // A first program in Java
3
4 public class Welcome1 {
5   public static void main( String args[] )
6   {
7     System.out.println("Welcome to Java Programming!");
8   }
9 }
```

```
Welcome to Java Programming!
```

A Simple Program: Printing a Line of Text

```
1 // Welcome1.java
```

- `//` indicates the remainder of the line is a comment
  - Comments are ignored by the compiler
  - Use comments to document and describe code
- Can also use multiple line comments: `/* ... */`

```
/* This is a multiple
   line comment. It can
   be split over many lines */
```

```
2 // A first program in Java
```

- Note: line numbers are not part of the program; they are added for our reference

A Simple Program: Printing a Line of Text

```
3
```

- A blank line
  - Blank lines and spaces make a program more readable
  - Blank lines, spaces, and tabs are known as *whitespace characters*, and are ignored by the compiler

```
4 public class Welcome1 {
```

- Begins a class definition for class **`Welcome1`**
  - Every Java program has at least one user-defined class
  - **`class`** keyword immediately followed by class name
    - Keyword: words reserved for use by Java
  - Naming classes: capitalize every word
    - **`SampleClassName`**

A Simple Program: Printing a Line of Text

```
4  public class Welcome1 {
```

- Name of class called *identifier*
  - Series of characters consisting of letters, digits, underscores (**_**) and dollar signs (**$**)
  - Does not begin with a digit
  - Contains no spaces
  - Examples: **Welcome1**, **$value**, **_value**, **button7**
  - **7button** is invalid
  - Case sensitive (capitalization matters)
    - **a1** and **A1** are different

A Simple Program: Printing a Line of Text

```
4  public class Welcome1 {
```

- Saving files
  - File name is class name and **`.java`** extension
  - **`Welcome1.java`**
- Left brace
  - Begins body of every class
  - Right brace ends definition (line **`9`**)

```
5     public static void main( String args[] )
```

- Part of every Java application
  - Applications begin executing at **`main`**
    - Parenthesis indicate **`main`** is a method
    - Java applications contain one or more methods

A Simple Program: Printing a Line of Text

```
5     public static void main( String args[] )
```

- Exactly one method must be called **`main`**

- Methods can perform tasks and return information
  - **`void`** means main returns no information
  - For now, mimic **`main`**'s first line

```
6     {
```

- Left brace begins body of method definition
  - Ended by right brace

A Simple Program: Printing a Line of Text

```
7        System.out.println( "Welcome to Java Programming!" );
```

- Instructs computer to perform an action
  - Prints string of characters between double quotes
    - String - series characters inside double quotes
  - White spaces in strings are not ignored by compiler
- **`System.out`** - standard output object
  - Allows java to print to command window (i.e., MS-DOS prompt)
- Method **`System.out.println`** displays a line of text
  - Argument inside parenthesis
- Entire line known as a statement
  - All statements must end with a semicolon - “`;`”

A Simple Program: Printing a Line of Text

1) **System: I**t is the name of standard class that contains objects that encapsulates the standard **I/O** devices of your system. It is contained in the **package** java.lang. Since java.lang package is imported in every java program by default, therefore **java.lang package** is the only package in Java API which doesnot require an import declaration.

2) **out:** The object out represents output stream (i.e Command window) and is the static data member of the class **system**. So note here **System.out** (System - Class & out - static object i.e why its simply referred by classname and we need not to create any object).

3) **println:** The println() is **method** of out object that takes the text string as an argument and displays it to the standard output i.e. *on monitor screen*.

System - Class

out - static Object

println() - method

A Simple Program: Printing a Line of Text

```
8     }
```

- Ends method definition

```
9  }
```

- Ends class definition
- Some programmers add comments to keep track of ending braces
- Lines 8 and 9 could be rewritten as:

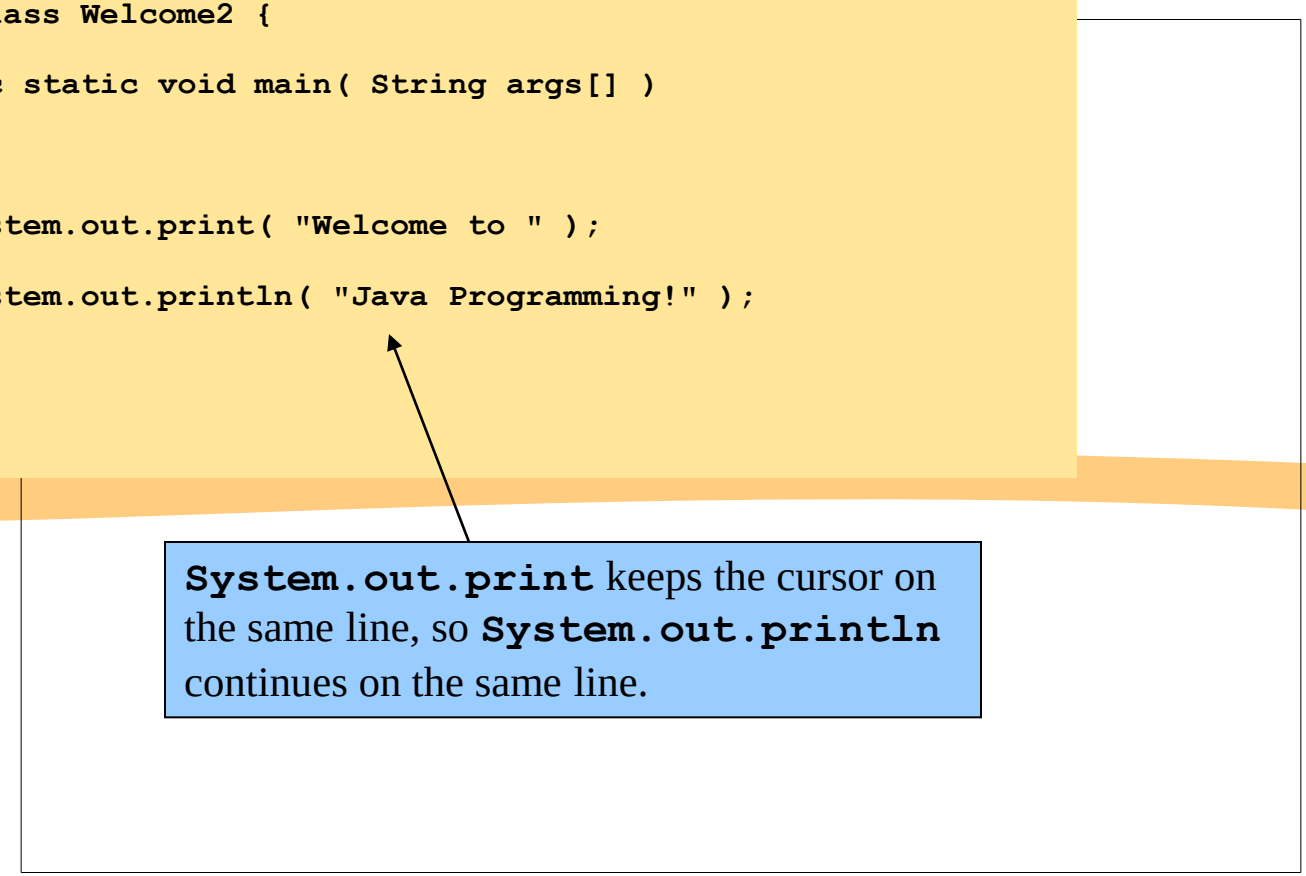
```
8     }  // end of method main()
```

```
9  }     // end of class Welcome1
```

A Simple Program: Printing a Line of Text

- Compiling a program
  - Open a command window, go to directory where program is stored
  - Type **`javac Welcome1.java`**
  - If there are no errors, file **`Welcome1.class`** is created
    - Contains Java bytecodes that represent application
    - Bytecodes passed to Java interpreter
- Executing a program
  - Type **`java Welcome1`**
    - Launches interpreter to load **`.class`** file for class **`Welcome1`**
    - **`.class`** extension omitted from command
  - Interpreter calls method **`main`**

```
1  // Fig. 2.3: Welcome2.java
2  // Printing a line with multiple statements
3
4  public class Welcome2 {
5     public static void main( String args[] )
6     {
7        System.out.print( "Welcome to " );
8        System.out.println( "Java Programming!" );
9     }
10 }
```



`System.out.print` keeps the cursor on the same line, so **`System.out.println`** continues on the same line.

```
Welcome to Java Programming!
```

A Simple Program: Printing a Line of Text

- Escape characters
  - Backslash (****)
  - Indicates that special characters are to be output
    - Backslash combined with a character makes an escape sequence
    - **\n** - newline
    - **\t** - tab
- Usage
  - Can use in **System.out.println** or **System.out.print** to create new lines
    - ```
      System.out.println(
      "Welcome\nto\nJava\nProgramming!" );
      ```
      ```

```
1  // Fig. 2.4: Welcome3.java
2  // Printing multiple lines with a single statement
3
4  public class Welcome3 {
5     public static void main( String args[] )
6     {
7        System.out.println( "Welcome\nto\nJava\nProgramming!" );
8     }
9  }
```

```
Welcome
to
Java
Programming!
```

Notice how a new line is output for each **\n** escape sequence.

A Simple Program: Printing a Line of Text

- Display
  - Although our first programs executed in the command window, most Java applications use windows or a dialog box
    - Netscape Communicator and Microsoft Internet Explorer execute in their own windows
  - Java has class **`JOptionPane`** that allows us to use dialog boxes
- Packages
  - Java has a set of predefined classes for us to use
  - Groups of related classes called *packages*
    - Group of all packages known as Java class library or Java applications programming interface (Java API)
  - **`JOptionPane`** is in the **`javax.swing`** package

A Simple Program: Printing a Line of Text

- Upcoming program
  - Application that uses dialog boxes - Explanation will come afterwards

```
1  // Fig. 2.6: Welcome4.java
2  // Printing multiple lines in a dialog box
3  import javax.swing.JOptionPane;  // import class JOptionPane
4
5  public class Welcome4 {
6     public static void main( String args[] )
7     {
8        JOptionPane.showMessageDialog(
9           null, "Welcome\nto\nJava\nProgramming!" );
10
11       System.exit( 0 );  // terminate the program
12    }
13 }
```

A Simple Program: Printing a Line of Text

- Lines 1-2: comments as before

```
3  import javax.swing.JOptionPane;  // import class JOptionPane
```

- **`import`** statements - locate the classes we intend to use
  - Tells compiler to load class **`JOptionPane`** from **`javax.swing`** package
    - This package contains many Graphical User Interface components

```
4
5  public class Welcome4 {
6     public static void main( String args[] )
```

- Lines 4-7: Blank line, begin class **`Welcome4`** and **`main`**

A Simple Program: Printing a Line of Text

```
8         JOptionPane.showMessageDialog(
9            null, "Welcome\nto\nJava\nProgramming!" );
```

- Call method **`showMessageDialog`** of class **`JOptionPane`**
  - Requires two arguments
  - Multiple arguments separated by commas (`,`)
  - For now, first argument always **`null`**
  - Second argument is string to display
- **`showMessageDialog`** is a **`static`** method of class **`JOptionPane`**
  - **`static`** methods called by using class name, dot (`.`) then method name
- All statements end with `;`
  - A single statement can therefore span multiple lines
  - Cannot split a statement in the middle of identifier or string

A Simple Program: Printing a Line of Text

```
8         JOptionPane.showMessageDialog(
9            null, "Welcome\nto\nJava\nProgramming!" );
```

- Executing lines 8 and 9 displays the dialog box shown below

  - Automatically includes an **OK** button
    - Hides or dismisses dialog box
  - Title bar has string **Message**

A Simple Program: Printing a Line of Text

```
11          System.exit( 0 );  // terminate the program
```

- Calls **`static`** method **`exit`** of class **`System`**
  - Terminates application
    - Use with any application displaying a GUI
  - Because method is **`static`**, needs class name and dot (**`.`**)
  - Identifiers starting with capital letters are usually class names
- Argument of **`0`** means application ended successfully
  - Non-zero usually means an error occurred
- Class **`System`** part of package **`java.lang`**
  - No **`import`** statement needed
  - **`java.lang`** automatically imported in every Java program
- Lines 12-13: End **`Welcome4`** and **`main`**

```
1  // Fig. 2.6: Welcome4.java
2  // Printing multiple lines in a dialog box
3  import javax.swing.JOptionPane;  // import class JOptionPane
4
5  public class Welcome4 {
6     public static void main( String args[] )
7     {
8        JOptionPane.showMessageDialog(
9           null, "Welcome\nto\nJava\nProgramming!" );
10
11       System.exit( 0 );  // terminate the program
12    }
13 }
```

A Simple Program: Printing a Line of Text

- Public static void showMessageDialog (Component parentComponent, Object message) throws HeadlessException

Brings up an information-message dialog titled "Message".

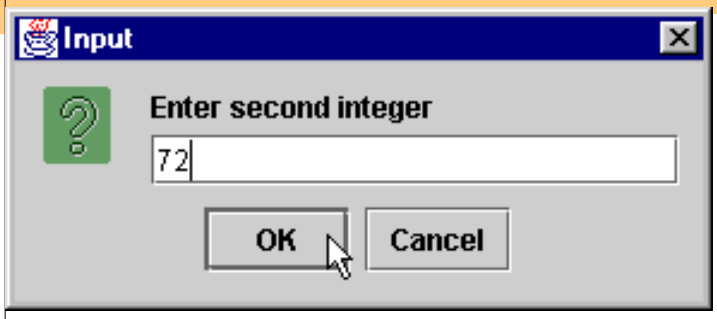
- Parameters: parentComponent - determines the Frame in which the dialog is displayed; if null, or if the parentComponent has no Frame, a default Frame is used
- message - the Object to display
- Throws: HeadlessException - if Graphics Environment is Headless returns true

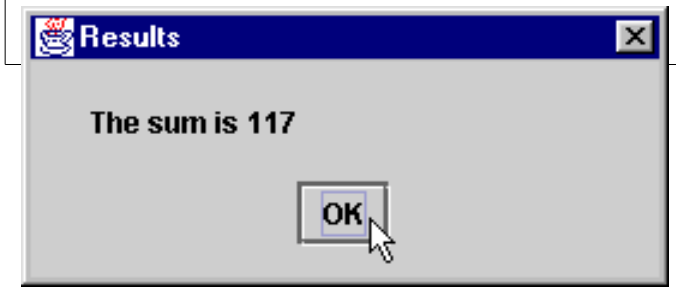
Another Java Application: Adding Integers

- Upcoming program
  - Use input dialogs to input two values from user
  - Use message dialog to display sum of the two values

```
1  // Fig. 2.8: Addition.java
2  // An addition program
3
4  import javax.swing.JOptionPane;  // import class JOptionPane
5
6  public class Addition {
7     public static void main( String args[] )
8     {
9        String firstNumber,   // first string entered by user
10              secondNumber;  // second string entered by user
11       int number1,          // first number to add
12           number2,          // second number to add
13           sum;              // sum of number1 and number2
14
15       // read in first number from user as a string
16       firstNumber =
17          JOptionPane.showInputDialog( "Enter first integer" );
18
19       // read in second number from user as a string
20       secondNumber =
21          JOptionPane.showInputDialog( "Enter second integer" );
22
23       // convert numbers from type String to type int
24       number1 = Integer.parseInt( firstNumber );
25       number2 = Integer.parseInt( secondNumber );
26
27       // add the numbers
28       sum = number1 + number2;
29
30       // display the results
```

```
31         JOptionPane.showMessageDialog(
32            null, "The sum is " + sum, "Results",
33            JOptionPane.PLAIN_MESSAGE );
34
35         System.exit( 0 );   // terminate the program
36      }
37   }
```





Another Java Application: Adding Integers

- Lines 1-2: Comments

```
4 import javax.swing.JOptionPane;  // import class JOptionPane
```

- Specifies location of **`JOptionPane`** for use in the program

```
6 public class Addition {
```

- Begins **`public`** class **`Addition`**
  - Recall that file name must be **`Addition.java`**
- Lines 7-8: **`main`**

```
 9        String firstNumber,   // first string entered by user
10               secondNumber;  // second string entered by user
```

  - **`firstNumber`** and **`secondNumber`** are variables

Another Java Application: Adding Integers

```
9         String firstNumber,   // first string entered by user
10                secondNumber;  // second string entered by user
```

- Variables: location in memory that can store a value
  - Must be declared with a name and data type before use
  - **`firstNumber`** and **`secondNumber`** are of data type **`String`** (**`java.lang`**), so they will hold strings
  - Variable name: any valid identifier
  - Declarations end with semicolons **`;`**
    - Can declare multiple variables of the same type at a time
    - Use a comma separated list
- Programmers often add comments to describe purpose of variables

Another Java Application: Adding Integers

```
11          int number1,            // first number to add
12              number2,            // second number to add
13              sum;                // sum of number1 and number2
```

- Declares variables **`number1`**, **`number2`**, and **`sum`** of type **`int`**
  - **`int`** can hold integer values (whole numbers): i.e., **`0`**, **`-4`**, **`97`**
  - Data types **`float`** and **`double`** can hold decimal numbers
  - Data type **`char`** can hold a single character
  - Known as primitive data types - more in Chapter 4

Another Java Application: Adding Integers

```
15          // read in first number from user as a string
16          firstNumber =
17             JOptionPane.showInputDialog( "Enter first integer" );
```

- Reads a **`String`** from the user, representing the first number to be added
  - Method **`JOptionPane.showInputDialog`** displays the following:

  - Message called a prompt - directs user to perform an action
  - Argument appears as prompt text
  - If wrong type of data entered (i.e. non-integer) then error occurs

Another Java Application: Adding Integers

```
15        // read in first number from user as a string
16        firstNumber =
17           JOptionPane.showInputDialog( "Enter first integer" );
```

- Result of call to **`showInputDialog`** (a **`String`** with the user input) given to **`firstNumber`** with the assignment operator **`=`**
  - Assignment statement
  - **`=`** binary operator - takes two *operands*
    - Expression on right evaluated and assigned to variable on left
  - Read as: **`firstNumber`** gets value of **`JOptionPane.showInputDialog( "Enter first integer" )`**

Another Java Application: Adding Integers

```
19        // read in second number from user as a string
20        secondNumber =
21           JOptionPane.showInputDialog( "Enter second integer" );
```

- Similar to previous statement
  - Assigns variable **`secondNumber`** to second integer input

```
23        // convert numbers from type String to type int
24        number1 = Integer.parseInt( firstNumber );
25        number2 = Integer.parseInt( secondNumber );
```

- Method **`Integer.parseInt`**
  - Converts its **`String`** argument into an integer (type **`int`**)
    - Class **`Integer`** in **`java.lang`**
  - Integer returned by **`Integer.parseInt`** is assigned to variable **`number1`** (line 24)
    - Remember that **`number1`** was declared as type **`int`**
  - Line 25 similar

Another Java Application: Adding Integers

```
27          // add the numbers
28          sum = number1 + number2;
```

- Assignment statement
  - First calculates sum of **`number1`** and **`number2`** (right hand side)
  - Next, uses assignment operator **`=`** to assign result to variable **`sum`**
  - Read as: **`sum`** gets the value of **`number1 + number2`**

Another Java Application: Adding Integers

```
31          JOptionPane.showMessageDialog(
32             null, "The sum is " + sum, "Results",
33             JOptionPane.PLAIN_MESSAGE );
```

- Uses **`showMessageDialog`** to display results
- **`"The sum is " + sum`**
  - Uses the operator + to "add" the string literal **`"The sum is"`** and **`sum`**
  - Allows concatenation of a **`String`** and another data type
    - Results in a new string
  - If **`sum`** contains **`117`**, then **`"The sum is " + sum`** results in the new string **`"The sum is 117"`**
  - Note the space in **`"The sum is "`**
  - More on strings in Chapter 10

Another Java Application: Adding Integers

```
31        JOptionPane.showMessageDialog(
32           null, "The sum is " + sum, "Results",
33           JOptionPane.PLAIN_MESSAGE );
```

- Requires four arguments (instead of two as before)
- First argument: **`null`** for now
- Second: message to display
- Third: string to display in title bar
- Fourth: value indicating type of message dialog
  - **`JOptionPane.PLAIN_MESSAGE`** indicates no icon
  - **`JOptionPane.ERROR_MESSAGE`**
  - **`JOptionPane.INFORMATION_MESSAGE`**
  - **`JOptionPane.WARNING_MESSAGE`**
  - **`JOptionPane.QUESTION_MESSAGE`**

End of Chapter 2

Questions?