Objectives of Lecture 9
The structure of a Java Program

- Understand the structure of a Java program and the different classes that form a program.
- Get an introduction to methods and invocation of methods by sending message expressions.
- Comprehend the relationship between program, classes and methods.
- Find out how applications and applets are launched.

Outline of Lecture 9

- Program
- Classes
- Methods
- Method dispatch
- Launching an application
- Launching an applet

The Structure of a Java Program

- There are four major structural components of Java programs
  - the program itself
  - classes
  - methods
  - statements

A Java Program - a Set of Classes

- A Java program consists of one or more classes.

Course Content

- Introduction
- Objects
- Method
- Tracing Programs
- Object State
- Sharing resources
- Selection
- Repetition
- Vectors
- Testing/Debugging
- Arrays
- Searching
- Files I/O
- Sorting
- Inheritance
- Recursion
Outline of Lecture 9

- Program
- Classes
- Methods
- Method dispatch
- Launching an application
- Launching an applet

Syntax for a Java Class

```java
public class Adventure {
    /* An instance of this class is an arithmetic adventure game where an adventurer navigates rooms that contain treasure chests that are opened by correctly answering arithmetic problems. */
    class comment
    ...
    class end delimiter
}
```

Outline of Lecture 9

- Program
- Classes
- Methods
- Method dispatch
- Launching an application
- Launching an applet

A Java Class - a Set of Methods

- The body of each Java class includes a set of methods.
- A method is some code that performs a single task.

Two Kinds of Methods

- There are two kinds of methods in Java.
- An instance method implements a message that is sent to an instance of the class.
- A static method implements a task that is independent of any particular object.
- In either case, some code is run and (optionally) a result is returned.

Syntax for a Java Method

```java
public static void main(String args[]) {
    /* Starting point for a program. */
    method start delimiter
    body of the method goes here
    method end delimiter
}
```
A Java Method - Statements

- The body of a method includes a sequence of statements.

Java Statements

- There are many kinds of Java statements.
- Each statement ends with a semi-colon.
- We have already seen four kinds of statements:
  - variable declaration
  - import
  - message expression
  - assignment statement

Outline of Lecture 9

- Program
- Classes
- Methods
  - Method dispatch
  - Launching an application
  - Launching an applet

Method Dispatch

- The association of messages to instance methods is called **method dispatch**.
- The class of the receiver object must contain an instance method with the same name as the message name.
- The class of each parameter in the parameter list of the method must match the class of each corresponding argument in the argument list of the message.

Method Dispatch Example 1

```
"Hello".toUpperCase();
```

```
String toUpperCase() {
  /*
   "HELLO"
  */
}
```

```
System.out.print("Hello");
```

```
public void print(String aString) {
  /*
   class of receiver is PrintStream
   message name is print
   one parameter class String
  */
}
```
Kinds of Java Programs

• Recall there are three kinds of programs:
  – Applications
  – Applets
  – Libraries
• The structure of all three kinds of programs are the same.
• However, each kind of program is launched differently.
• Libraries are never launched, they are just called by other programs.

Java Applications - launching

• In a Java application, one class is marked as the special “starting” class.
• When the Java application is launched by the interpreter, it invokes a static method called “main” in the start class.

Java Applications - main Protocol

• The start class must contain a static method for main with protocol:
  public static void main(String args[])

Outline of Lecture 9

• Program
• Classes
• Methods
• Method dispatch
• Launching an application
• Launching an applet

Java Applets - launching

• When the web browser reads a document that tells it to load an applet, it creates an instance of your applet subclass and sends it the instance message init().
Java Applets - init

- The *init()* message creates all of the graphical objects in the applet, like buttons and fields and puts them into your applet object.
- If you do not want to put any graphical objects in your applet, you do not need to implement an *init()* method in your applet subclass.

Java Applets - paint

- Whenever your applet must be displayed, the paint message is sent to your applet.
- For example, the paint message is sent after your applet is first initialized and any time the screen must be refreshed.
- The protocol for the paint message is:
  ```java
  public void paint(Graphics aGraphics);
  ```
- The paint method in your applet subclass must display any objects that you did not put in your applet with the *init()* method.

Objectives of Lecture 10

Implementing Classes - Methods

- Attempt to implement our first class by writing a collection of methods.

Outline of Lecture 10

- Restructuring the start class
- Self reference - this
- The return statement
- Adventure Version 2

The Start Class

- We have already implemented a class in our simple Java programs:
  ```java
  public class Adventure {
    /* Version 1
      This program is an arithmetic adventure...
      */
    
    However, we have not used this class for anything except to hold the static main() method that starts our program and contains all the code.
  }
  ```

The Program Object - Adventure

- Instead, we can restructure the code by creating multiple objects and methods.
- In the static main() method, we create an Adventure object and send it the play() message.
- The play() message is implemented by an instance method in the Adventure class.
Multiple Objects and Messages

- The problem is decomposed so that the play() method creates other objects and sends messages to them.
- This is a prototype for all application programs since they can all be structured the same way.

The new main() Method

- Create an instance of the start class, Adventure.
- Send it the play() message to play the game.

Program - Adventure 2.1

```java
import java.util.*;

public class Adventure {
    /* Version 2
    This program is an arithmetic adventure game ...
    */
    /* Constructors */
    public Adventure () {
        /* Initialize an Adventure by creating the appropriate objects. */
    }
}
```

Program - Adventure 2.2

```java
/* Main program */
public static void main(String args[]) {
    Adventure game;
    game = new Adventure();
    game.play();
}
```

Outline of Lecture 10

- Restructuring the start class
- Self reference - this
- The return statement
- Adventure Version 2

Self-Referencing

- Inside of a method, we often need to send a message to the receiver of the current message.
- That is, we need an object reference to the current object.
The Java Variable called *this*

- In a natural language, self referencing is done using the word *me* or *I*.
- In Java, the word *this* is used for self reference.
- If the variable *this* appears in a method, it refers to the the receiver object of that method.

```java
private void farewell(String userName, Integer tokenCount) { 
    /* Say farewell to the user with the given name and report the given count of tokens earned. */
    System.out.print("Congratulations ");
    System.out.print(userName);
    System.out.print(" you have left the game with ");
    System.out.print(tokenCount);
    System.out.println(" tokens.");
}
```

Program - Adventure 2.3

```java
/* Private Instance Methods */
private void play() {
    /* Play the Adventure game. */
    String name;
    Integer tokens;
    name = this.greeting();
    tokens = this.enterRoom(name);
    this.farewell(name, tokens);
}
```

The Return Statement

- A **return statement** is used in a method to return the result object or value.
- The syntax of the return statement is:
  
  `<return statement> ::= return <reference>`

- The class of the object or value reference that is returned must match the return type specified in the method signature.

Outline of Lecture 10

- Restructuring the start class
- Self reference - *this*
  - The return statement
  - Adventure Version 2
Program - Adventure 2.5

```java
private String greeting() {
  /**
   * Greet the user and answer a String that represents
   * the player’s name.
   */
  String playerName;
  System.out.println("Welcome to the Arithmetic Adventure game.");
  System.out.println("The date is ");
  System.out.println(new Date());
  System.out.println("What is your name?");
  playerName = Keyboard.in.readString();
  System.out.println("Well ");
  System.out.println(playerName);
  System.out.println(" , after a day of hiking you spot a silver cube.");
  System.out.println("The cube appears to be about 5 meters on each side.");
  System.out.println("You find a green door, open it and enter.");
  System.out.println("The door closes behind you with a soft whir and disappears.");
  System.out.println("There is a feel of mathematical magic in the air.");
  Keyboard.in.pause();
  return playerName;
}
```

Program - Adventure 2.6

```java
System.out.println("Well ");
System.out.println(playerName);
System.out.println("You find a green door, open it and enter.");
System.out.println("There is a feel of mathematical magic in the air.");
Keyboard.in.pause();
return playerName;
```

Program - Adventure 2.7

```java
private Integer enterRoom(String theName) {
  /**
   * The user with the given name has entered the
   * first room. After the adventure is done, return the
   * number of tokens obtained during the game.
   */
  Integer myTokens;
  System.out.println("How many tokens would you like, ");
  System.out.println(theName);
  myTokens = Keyboard.in.readInteger();
  return myTokens;
}
```

Adventure 2 Output

The Big Picture

What happens in `play()`?
What happens between `play()` and `greeting()`?

What happens between `play()` and `enterRoom()`?

What happens between `play()` and `farewell()`?