INTRODUCTION TO OBJECT-ORIENTED PROGRAMMING

CIS*2430 (Fall 2010)

Contact Information

- Instructor: Fei Song
  - Email: fsong@uoguelph.ca
  - Office: Reynolds 215, ext. 58067
  - Office hours: Mon/Wed/Fri.: 3:30 – 4:30 pm at the instructor's office
- TA's: Will Darling and Tao Xu
  - Email: ta2430@cis.uoguelph.ca
  - Responsibilities: teach labs, mark assignments and exams, and maintain email and forum discussions.

Evaluation

- Programming (50%)
  - Four assignments @ 10% each
  - Five lab assignments @ 2% each
- Exams (50%)
  - Two midterms @ 10% each
  - One lab exam @ 5%
  - One final exam @ 25%

- Learning by doing is the best way to learn a programming language and the related concepts.

Policies

- Come to the class regularly: the textbook and lecture notes can't replace question-answering and in-class discussions

- Labs are required:
  - All labs will take place in Thornbrough 1319
  - Three scheduled sessions:

    Section 1: Mon, 2:30 – 4:20 pm

    Section 2: Tue, 1:00 – 2:50 pm

    Section 3: Thu, 1:00 – 2:50 pm.
  - All lab assignments are to be done in the lab.

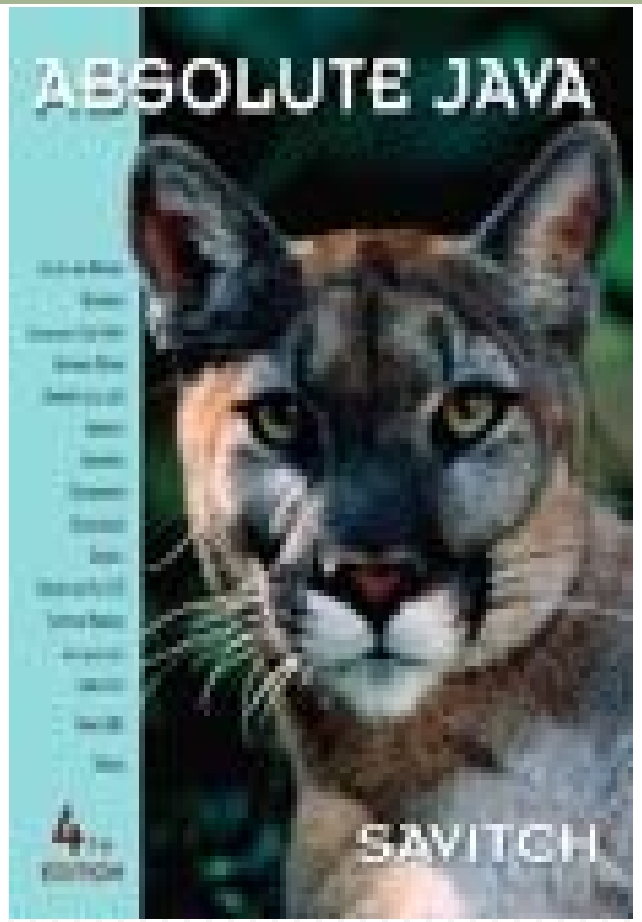
Policies (continued)

- Late assignments are not accepted and any requests for re-marking need to be submitted within 5 business days
- Do your own work individually. Collaboration and code reuse is encouraged, but must be documented clearly and appropriately
- Emergencies do happen and you need to contact me as soon as you have a problem - don't wait until it is a disaster.

Textbook

- Walter Savitch. *Absolute Java*. Fourth Edition. Addison-Wesley, 2009 (ISBN: 0-13-608382-X)

- Additional course materials are available at the course website: moodle.cis.uoguelph.ca



List of Topics

- OOP introduction
- Class design in terms of variables and methods
- Information hiding and encapsulation
- Inheritance, polymorphism, and overloading
- Data structures such as arrays, ArrayLists, and HashMaps
- Containers and iterators
- Exception handling and event-driven programming
- Swings and GUI's
- OO Analysis and design techniques

Advice from Previous Classes

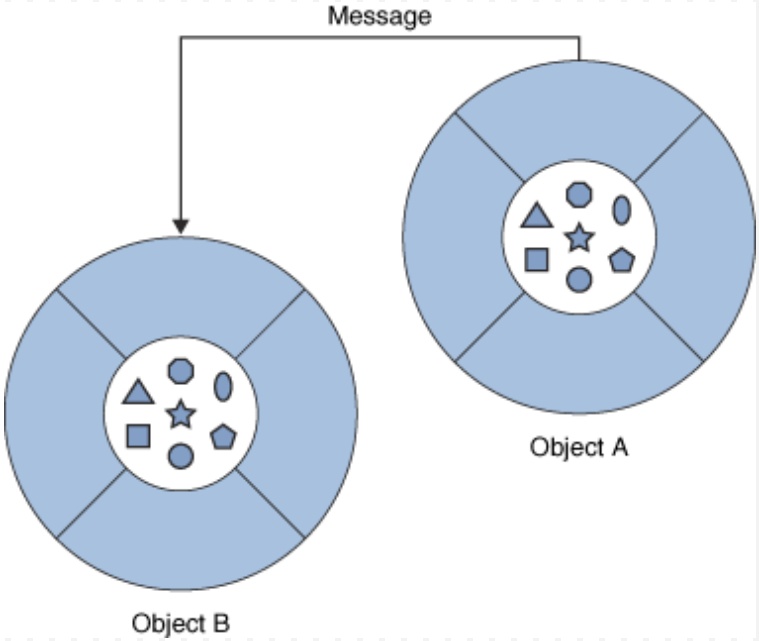
- Start assignments early: big tasks become small tasks and you will learn more and enjoy yourself more
- Don’t leave assignments to the last minute: small tasks become big tasks and you will become less productive under pressure
- Buy the textbook: extensive coverage of additional examples and exercises
- Don't write a line of code until you spend some quality time with pen and paper in hand, thinking about how best to solve the problem
- Chances are you don't know how to test and debug well. Read ahead and learn it because you can bring programming time down.

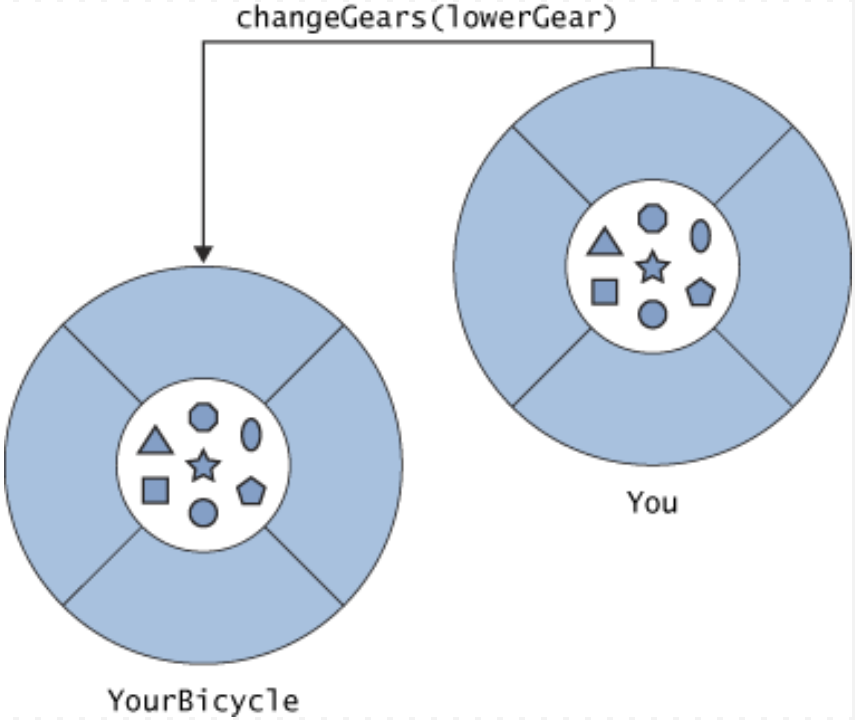
Introduction to OPP

- Objects
- Programming Paradigms
- Java
  - History
  - Basics
  - Console programming

Objects?

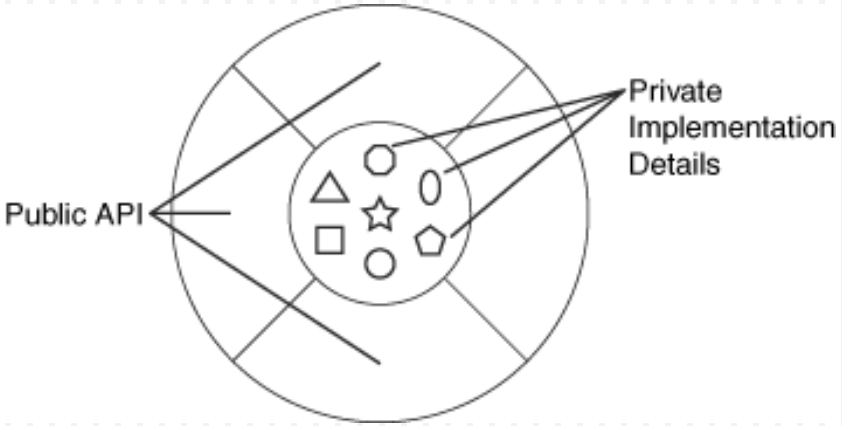
Message Passing

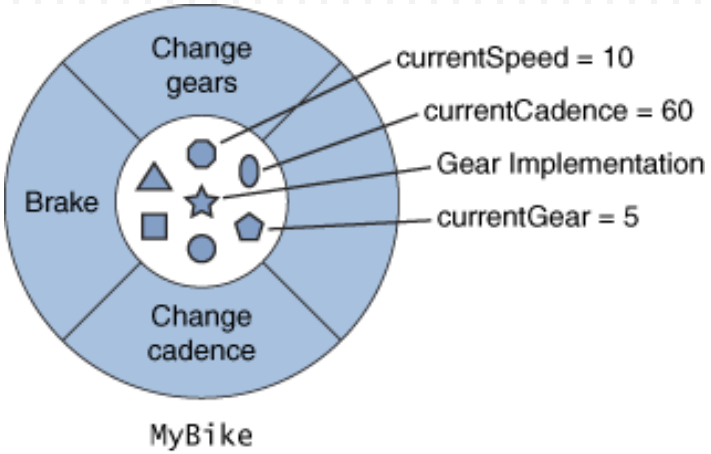


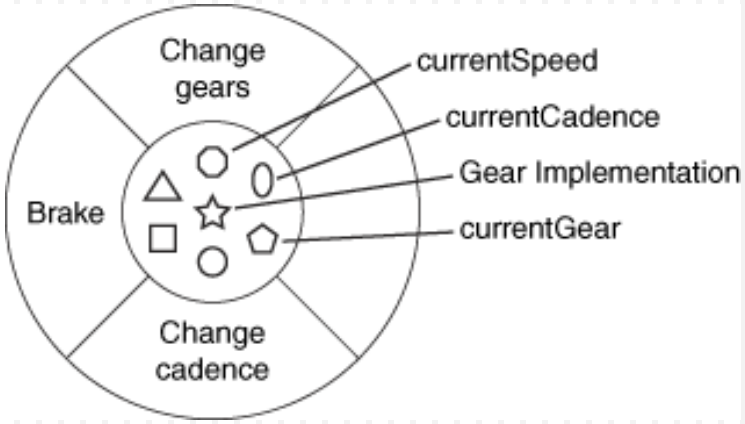


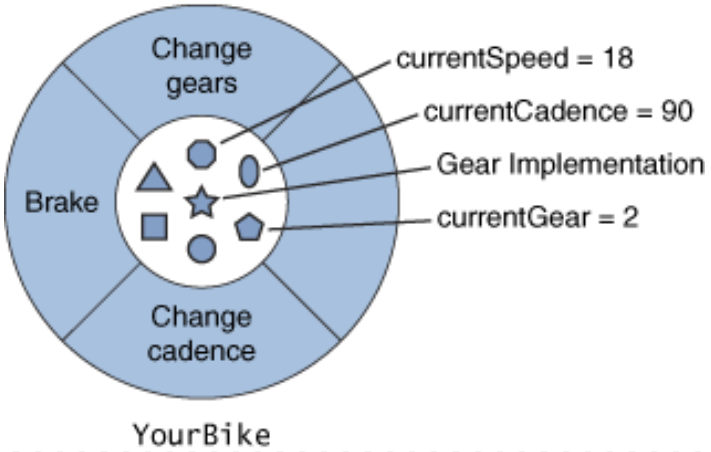
http://java.sun.com/docs/books/tutorial/java/concepts/message.html

Class and its Objects









http://java.sun.com/docs/books/tutorial/java/concepts/class.html

Programming Paradigms

Procedural or Imperative	Details of how computation happens	i := 0; sum := 0;
		while (i < n) do
		i := i + 1;
		sum := sum + 1
		end;

Functional	Describe what but not how	func sum(n:int) : int;
		if n = 0
		then 0
		else n + sum(n-1)
		end;

Logic	Describe knowledge and answers queries.	edge(a,b). edge(a,c). edge(c,a). path(X,X). path(X,Y) :- edge(X,Y). path(X,Y) :- edge(X,Z), path(Z,Y).
OO	Abstract Data Types and services	Price moreCost = new Price(); moreCost.increasePrice(5);

http://www.cs.wright.edu/~tkprasad/courses/cs784/L5Pdm.ppt

Procedural vs Object-Oriented

Programs = Data + Algorithms

- Emphasis on procedural abstraction.
- Top-down design;
  Step-wise refinement.
- Suited for programming in the small.

- Emphasis on data abstraction.
- Bottom-up design;
  Reusable libraries.
- Suited for programming in the large.

http://www.cs.wright.edu/~tkprasad/courses/cs784/L5Pdm.ppt

Introduction to Java

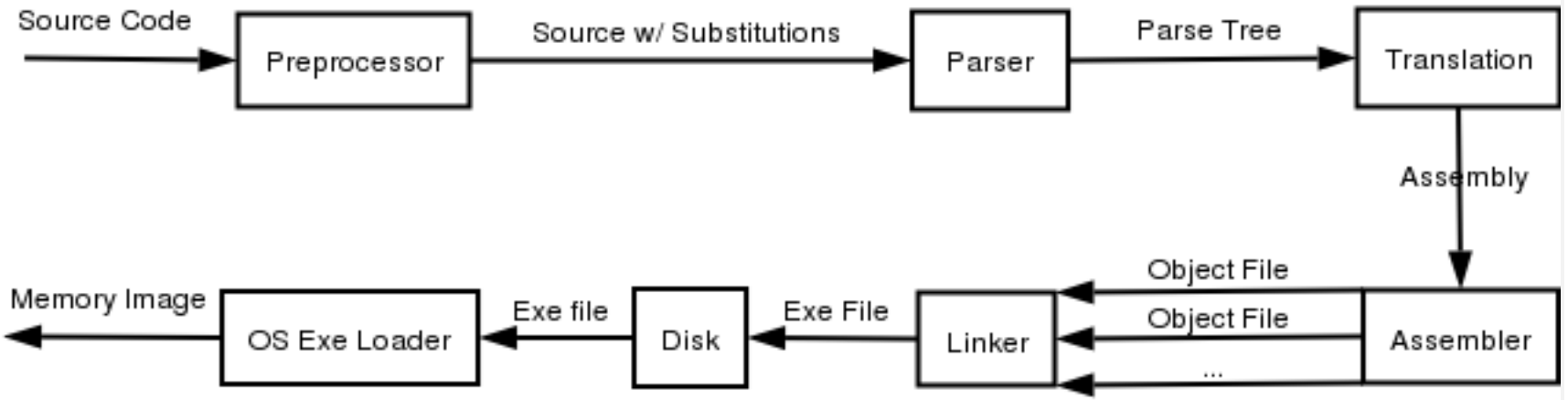
- Most people are familiar with Java as an Internet programming language (e.g., Applets)

- We will study Java as a general purpose programming language (e.g., Applications)
  - The syntax of expressions and assignments is similar to that of most other high-level languages
  - Details concerning the handling of objects and console input/output are new.

Why Java?

- Java is a pure object-oriented programming language
- Has built-in supports for Internet, GUI's, concurrent, and network programming
- Has evolved into a programming platform (e.g., J2EE)

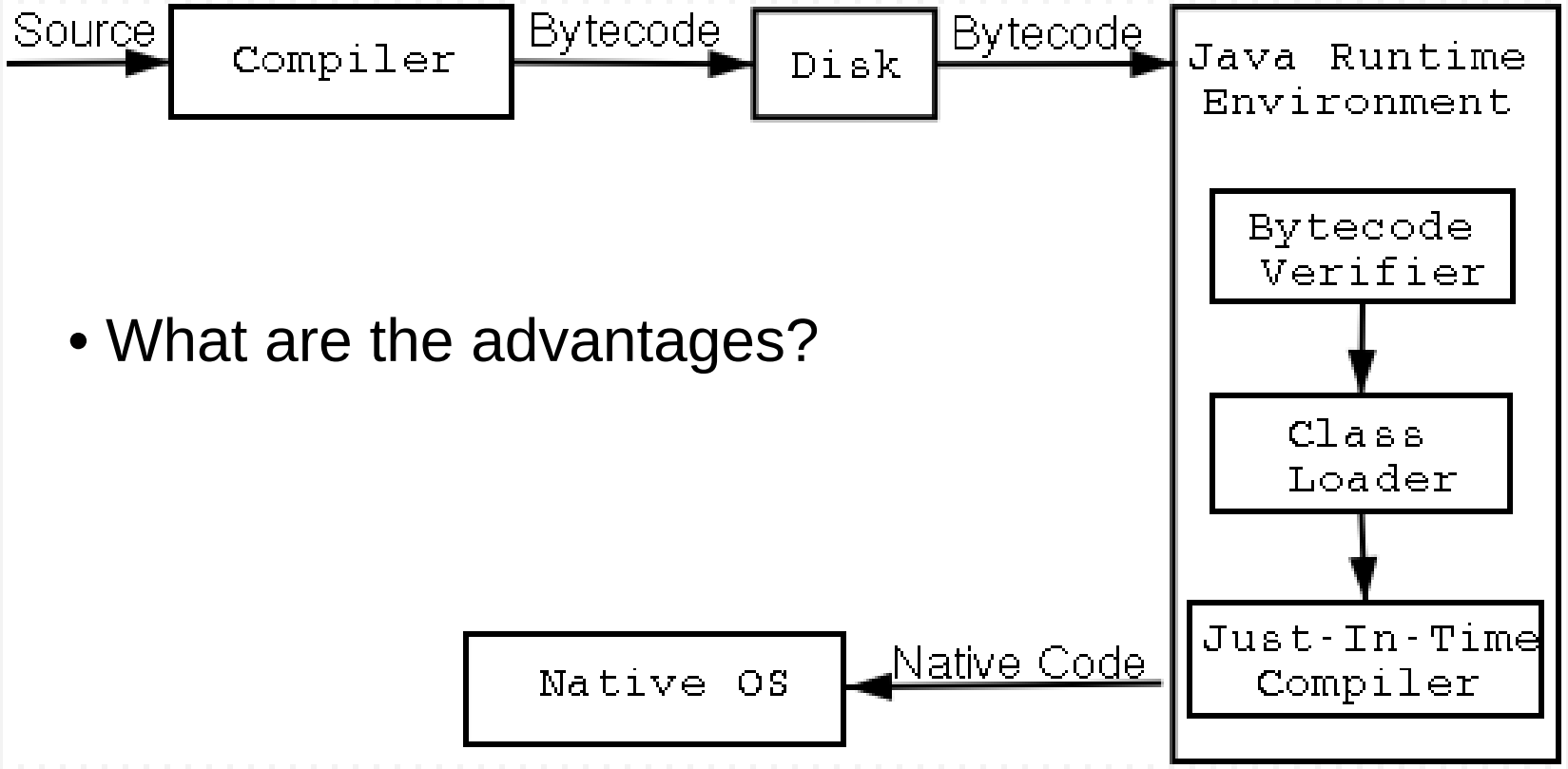
Major Compilation Steps



- What are the roles of a compiler?

http://www.acm.uiuc.edu/sigmil/RevEng/ch02.html

Java Compilation



- What are the advantages?

http://www.acm.uiuc.edu/sigmil/RevEng/ch02.html

Debugging and Errors

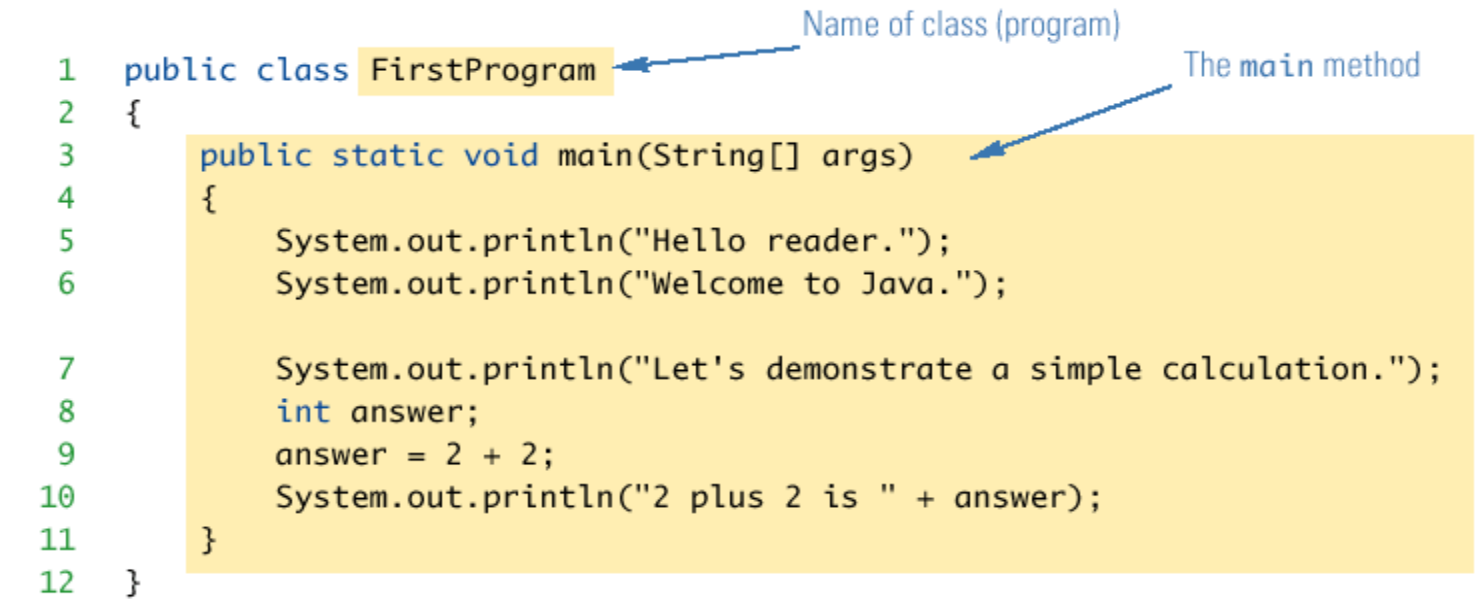
- The process of eliminating bugs is called *debugging*
  - *Bug: a mistake in a program*
- *Syntax error:* a grammatical mistake in a program
  - The compiler can detect these errors, and will output an error message, explaining what it thinks the error is, and where it thinks the error is located.
  - Syntax errors are relatively easy to fix because they are explicit.

Debugging and Errors

- *Run-time error:* an error that is not detected until a program is run
  - The compiler cannot detect these errors: an error message is not generated after compilation, but after execution
- *Logic error:* a mistake in the underlying algorithm for a program
  - *The compiler cannot detect these errors, and no error message is generated after compilation or execution, but the program does not do what it is supposed to do.*

Sample Application

Display 1.1 A Sample Java Program




```
1  public class FirstProgram
2  {
3      public static void main(String[] args)
4      {
5          System.out.println("Hello reader.");
6          System.out.println("Welcome to Java.");

7          System.out.println("Let's demonstrate a simple calculation.");
8          int answer;
9          answer = 2 + 2;
10         System.out.println("2 plus 2 is " + answer);
11     }
12 }
```

SAMPLE DIALOGUE 1

```
Hello reader.
Welcome to Java.
Let's demonstrate a simple calculation.
2 plus 2 is 4
```

Java Applications

- An application consists one or more classes and one of them should have a “main” method.
- Each class is saved into a file whose name is the same as the classname, e.g., FirstProgram.java.
- Compiling a program: “javac FirstProgram.java” will produce a byte-code file: FirstProgram.class.
- Running a program: “java FirstProgram” (no extension) will activate the “main” method in the given program.

Class Loader

- Java programs are divided into smaller parts called *classes*
  - Each class definition is normally stored in a separate file and compiled separately
- *Class Loader*: A program that connects the byte-code of the classes needed to run a Java program
  - In other programming languages, the corresponding program is called a *linker*.

Identifiers

- Java identifiers must not start with a digit, and all the characters must be letters, digits, or the underscore symbol
- Java identifiers can theoretically be of any length
- Java is case-sensitive: `Rate`, `rate`, and `RATE` are different names.

Identifiers

- Keywords and reserved words: don't use them to name something else

```
public    class    void    static
```

- Predefined identifiers in libraries required by the Java language standard names: can be redefined but should be avoided

```
System    String    println
```

Naming Conventions

- Start the names of variables, methods, and objects with a lowercase letter, indicate "word" boundaries with an uppercase letter, and restrict the remaining characters to digits and lowercase letters

  `topSpeed` `bankRate1` `timeOfArrival`

- Start the names of classes with an uppercase letter

  `FirstProgram` `MyClass` `String`

Variable Declarations

- Every variable must be *declared* before its use:
  - A variable declaration tells the compiler what kind of data (type) will be stored in the variable
  - Variables are typically declared just before they are used or at the start of a block (indicated by a pair of braces **{}**)

```
int numberOfBeans;
double oneWeight, totalWeight;
```

Primitive Data Types

Display 1.2 Primitive Types

TYPE NAME	KIND OF VALUE	MEMORY USED	SIZE RANGE
`boolean`	`true` or `false`	1 byte	not applicable
`char`	single character (Unicode)	2 bytes	all Unicode characters
`byte`	integer	1 byte	−128 to 127
`short`	integer	2 bytes	−32768 to 32767
`int`	integer	4 bytes	−2147483648 to 2147483647
`long`	integer	8 bytes	−9223372036854775808 to 9223372036854775807
`float`	floating-point number	4 bytes	$-3.40282347 \times 10^{+38}$ to $-1.40239846 \times 10^{-45}$
`double`	floating-point number	8 bytes	$\pm 1.76769313486231570 \times 10^{+308}$ to $\pm 4.94065645841246544 \times 10^{-324}$

Initializing Variables

- The declaration of a variable can be combined with its initialization via an assignment statement:

```
int count = 0;
double distance = 55 * .5;
char grade = 'A';
```

- In certain cases uninitialized variables are given default values:
  - It is best not to rely on this
  - Explicitly initialized variables have the added benefit of improving program clarity.

Shorthand Assignment Statements

Example:	Equivalent To:
`count += 2;`	`count = count + 2;`
`sum -= discount;`	`sum = sum – discount;`
`bonus *= 2;`	`bonus = bonus * 2;`
`time /= rushFactor;`	`time = time / rushFactor;`
`change %= 100;`	`change = change % 100;`
`amount *= count1 + count2;`	`amount = amount * (count1 + count2);`

String Class

- There is no primitive type for strings in Java
- The class **`String`** is a predefined class used to store and process strings
- Objects of type **`String`** are made up of strings of characters:

```
String blessing = "Live long and prosper.";
```

Concatenation of Strings

- Use the + operator on two strings to get a longer string:
  - If `greeting` equals `"Hello "` and `javaClass` equals `"class"`, then `greeting + javaClass` equals `"Hello class"`
  - Any number of strings can be concatenated together

- When a string is combined with almost any other type of data, the result is a string:

`"The answer is " + 42` equals `"The answer is 42"`

String Methods

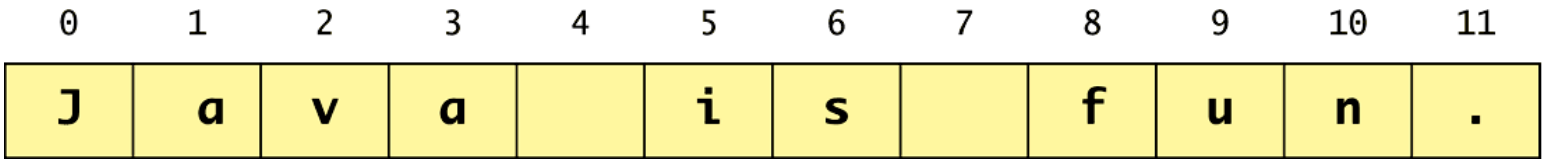
- The `String` class contains many useful methods :
  - A `String` method is called by writing a `String` object, a dot, the name of the method, and a pair of parentheses to enclose any arguments
  - If a `String` method returns a value, then it can be placed anywhere that a value of its type can be used

```
String greeting = "Hello";
int count = greeting.length();
System.out.println("Length is " +
   greeting.length());
```

String Indexes

Display 1.5 String Indexes

The 12 characters in the string `"Java is fun."` have indexes 0 through 11.



Notice that the blanks and the period count as characters in the string.

Some String Methods

Display 1.4 Some Methods in the Class `String`

`int length()`

Returns the length of the calling object (which is a string) as a value of type `int`.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.length()` returns `6`.

`boolean equals(`*Other_String*`)`

Returns `true` if the calling object string and the *Other_String* are equal. Otherwise, returns `false`.

EXAMPLE

After program executes `String greeting = "Hello";`
`greeting.equals("Hello")` returns `true`
`greeting.equals("Good-Bye")` returns `false`
`greeting.equals("hello")` returns `false`

Note that case matters. `"Hello"` and `"hello"` are not equal because one starts with an uppercase letter and the other starts with a lowercase letter.

(continued)

Some String Methods

Display 1.4 Some Methods in the Class `String`

`boolean equalsIgnoreCase(`*Other_String*`)`

Returns `true` if the calling object string and the *Other_String* are equal, considering uppercase and lowercase versions of a letter to be the same. Otherwise, returns `false`.

EXAMPLE

After program executes `String name = "mary!";`
`greeting.equalsIgnoreCase("Mary!")` returns `true`

`String toLowerCase()`

Returns a string with the same characters as the calling object string, but with all letter characters converted to lowercase.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.toLowerCase()` returns `"hi mary!"`.

(continued)

Some String Methods

Display 1.4 Some Methods in the Class `String`

`char charAt(`*Position*`)`

Returns the character in the calling object string at the *Position*. Positions are counted 0, 1, 2, etc.

EXAMPLE

After program executes `String greeting = "Hello!";`
`greeting.charAt(0)` returns `'H'`, and
`greeting.charAt(1)` returns `'e'`.

`String substring(`*Start*`)`

Returns the substring of the calling object string starting from *Start* through to the end of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2)` returns `"cdefG"`.

(continued)

Some String Methods

Display 1.4 **Some Methods in the Class `String`**

`String substring(`*Start*`, `*End*`)`

Returns the substring of the calling object string starting from position *Start* through, but not including, position *End* of the calling object. Positions are counted 0, 1, 2, etc. Be sure to notice that the character at position *Start* is included in the value returned, but the character at position *End* is not included.

EXAMPLE

After program executes `String sample = "AbcdefG";`
`sample.substring(2, 5)` returns `"cde"`.

`int indexOf(`*A_String*`)`

Returns the index (position) of the first occurrence of the string *A_String* in the calling object string. Positions are counted 0, 1, 2, etc. Returns `-1` if *A_String* is not found.

EXAMPLE

After program executes `String greeting = "Hi Mary!";`
`greeting.indexOf("Mary")` returns `3`, and
`greeting.indexOf("Sally")` returns `-1`.

(continued)

Coding Conventions

```
public class Pet
{
    private String name;
    private int age;        //in years
    private double weight;  //in pounds

    public String toString( )
    {
        return ("Name: " + name + " Age: " + age + " years"
                        + "\nWeight: " + weight + " pounds");
    }

    /**comment about the method here- for api documentation*/
    public Pet(String initialName, int initialAge, double initialWeight)
    {
        name = initialName;
        if ((initialAge < 0) || (initialWeight < 0))
        {
            System.out.println("Error: Negative age or weight.");
            System.exit(0);
        }
        else
        {
            age = initialAge;  //programmer only (and very few)
            weight = initialWeight;
        }
    }
...
```

Comments and a Named Constant

Display 1.8 Comments and a Named Constant

```
1  /**
2   Program to show interest on a sample account balance.
3   Author: Jane Q. Programmer.
4   E-mail Address: janeq@somemachine.etc.etc.
5   Last Changed: September 21, 2004.
6  */
7  public class ShowInterest
8  {
9      public static final double INTEREST_RATE = 2.5;

10     public static void main(String[] args)
11     {
12         double balance = 100;
13         double interest; //as a percent

14         interest = balance * (INTEREST_RATE/100.0);
15         System.out.println("On a balance of $" + balance);
16         System.out.println("you will earn interest of $"
17                                                 + interest);
18         System.out.println("All in just one short year.");
19     }
20
21 }
```

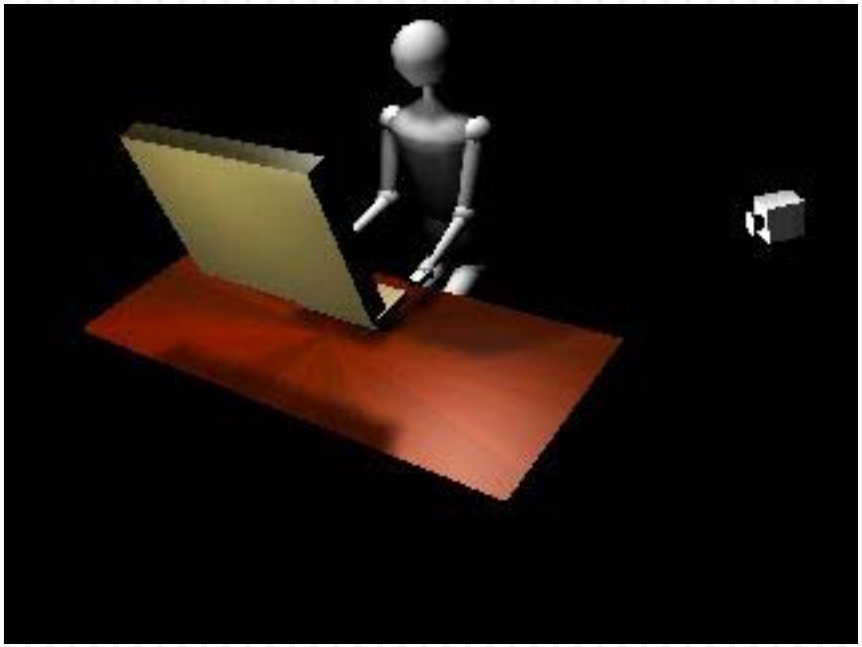
Although it would not be as clear, it is legal to place the definition of INTEREST_RATE here instead.

SAMPLE DIALOGUE

```
On a balance of $100.0
you will earn interest of $2.5
All in just one short year.
```

Communicating with Users

- Output
  - println, print, printf
- Input
  - Scanner class



http://www.cse.unsw.edu.au/~danielw/hci/

Formatted Output

Display 2.1 **Format Specifiers for `System.out.printf`**

CONVERSION CHARACTER	TYPE OF OUTPUT	EXAMPLES
d	Decimal (ordinary) integer	`%5d` `%d`
f	Fixed-point (everyday notation) floating point	`%6.2f` `%f`
e	E-notation floating point	`%8.3e` `%e`
g	General floating point (Java decides whether to use E-notation or not)	`%8.3g` `%g`
s	String	`%12s` `%s`
c	Character	`%2c` `%c`

Keyboard Input Demonstration

Display 2.6 Keyboard Input Demonstration

```
1  import java.util.Scanner;

2  public class ScannerDemo
3  {
4     public static void main(String[] args)
5     {
6        Scanner keyboard = new Scanner(System.in);

7        System.out.println("Enter the number of pods followed by");
8        System.out.println("the number of peas in a pod:");
9        int numberOfPods = keyboard.nextInt();
10       int peasPerPod = keyboard.nextInt();

11       int totalNumberOfPeas = numberOfPods*peasPerPod;

12       System.out.print(numberOfPods + " pods and ");
13       System.out.println(peasPerPod + " peas per pod.");
14       System.out.println("The total number of peas = "
15                                          + totalNumberOfPeas);
16    }
17 }
```

Makes the Scanner *class available to your program.*

Creates an object of the class Scanner *and names the object* keyboard.

Each reads one int *from the keyboard*

(continued)

Keyboard Input Demonstration

Display 2.6 Keyboard Input Demonstration

SAMPLE DIALOGUE 1

```
Enter the number of pods followed by
the number of peas in a pod:
22 10
22 pods and 10 peas per pod.
The total number of peas = 220
```

The numbers that are input must be separated by whitespace, such as one or more blanks.

SAMPLE DIALOGUE 2

```
Enter the number of pods followed by
the number of peas in a pod:
22
10
22 pods and 10 peas per pod.
The total number of peas = 220
```

A line break is also considered whitespace and can be used to separate the numbers typed in at the keyboard.

Another Keyboard Input Demonstration

Display 2.7 Another Keyboard Input Demonstration

```
1  import java.util.Scanner;

2  public class ScannerDemo2
3  {
4      public static void main(String[] args)
5      {
6          int n1, n2;
7          Scanner scannerObject = new Scanner(System.in);

8          System.out.println("Enter two whole numbers");
9          System.out.println("seperated by one or more spaces:");

10         n1 = scannerObject.nextInt();
11         n2 = scannerObject.nextInt();
12         System.out.println("You entered " + n1 + " and " + n2);

13         System.out.println("Next enter two numbers.");
14         System.out.println("Decimal points are allowed.");
```

Creates an object of the class `Scanner` *and names the object* `scannerObject`.

Reads one `int` *from the keyboard.*

(continued)

Another Keyboard Input Demonstration

Display 2.7 Another Keyboard Input Demonstration

```
15          double d1, d2;
16          d1 = scannerObject.nextDouble();
17          d2 = scannerObject.nextDouble();
18          System.out.println("You entered " + d1 + " and " + d2);

19          System.out.println("Next enter two words:");

20          String word1 = scannerObject.next();
21          String word2 = scannerObject.next();
22          System.out.println("You entered \"" +
23                                  word1 + "\" and \"" + word2 + "\"");

24          String junk = scannerObject.nextLine(); //To get rid of '\n'

25          System.out.println("Next enter a line of text:");
26          String line = scannerObject.nextLine();
27          System.out.println("You entered: \"" + line + "\"");
28      }
29  }
```

Reads one `double` *from the keyboard.*

Reads one word from the keyboard.

This line is explained in the Pitfall section "Dealing with the Line Terminator, `'\n'`*"*

Reads an entire line.

(continued)

Another Keyboard Input Demonstration

Display 2.7 Another Keyboard Input Demonstration

SAMPLE DIALOGUE

```
Enter two whole numbers
separated by one or more spaces:
  42   43
You entered 42 and 43
Next enter two numbers.
A decimal point is OK.
 9.99  57
You entered 9.99 and 57.0
Next enter two words:
jelly beans
You entered "jelly" and "beans"
Next enter a line of text:
Java flavored jelly beans are my favorite.
You entered "Java flavored jelly beans are my favorite."
```

Pitfall: Dealing with the Line Terminator, '\n'

- Method **`nextLine`** of class **`Scanner`** reads the remainder of a line of text starting wherever the last keyboard reading left off

- Given the code:

```
Scanner keyboard = new Scanner(System.in);
int n = keyboard.nextInt();
String s1 = keyboard.nextLine();
String s2 = keyboard.nextLine();
```

and the input:

```
2
Heads are better than
1 head.
```

What are the values of **`n`**, **`s1`**, and **`s2`**?