COP 3330, Spring 2013

Exam 1 Review

Instructor : Arup Ghosh
02-13-13

School of Electrical Engineering and Computer Science
University of Central Florida

Exam 1

- Monday, February 18.
- In class, 50 minutes long.
- No aids of any kind.
- Things to bring
  - Yourself
  - PEN

Things *not* on the exam

- Exceptions
- Containers (ArrayList, TreeSet, TreeMap)

Exam 1 format

- 2 sections – Total : 100 points
- 1st Section – Multiple Choice / TF (20*3 = 60)
- (Quiz 1 to 4?)

- 2nd Section – Free response (5*8 = 40)

The Java Virtual Machine

- Java does not compile source code (.java files) to native code.
- It is compiled to *bytecode* instead (.class files).
- Bytecode is executed on the Java Virtual Machine (JVM)
- Bytecode is platform-independent.
- So Java is *portable* – its programs can be run on any OS with a JVM without needing to be recompiled.

Memory management

- Unlike C, we do not have to allocate and free memory manually.
- Memory allocation is a simple use of the `new` operator.
- Memory is automatically freed by a process known as *garbage collection*.
- This eliminates problems like memory leaks.

Language Basics

- Variables
  - Primitives
  - Objects
- Control structures
  - if / else if / else, for, while, do-while, switch

Primitives

- Eight primitive data types (basically C-style variables)
  - `byte`
  - `int`
  - `short`
  - `long`
  - `float`
  - `double`
  - `char`
  - `boolean`

- Use them the way you would use C variables.

Language Basics

- Literals
  - boolean literals (true, false)
  - byte, short, int, long literals
  - float, double literals
  - char literals
  - String literals

Language Basics

- Expressions
  - Arithmetic Expressions
    - +, -, *, /, %
  - Assignment
    - =, +=, -=, *=, /=, %=
  - Comparison
    - ==, !=, <, >, <=, >=
  - Logical Expressions
    - &&, ||, !

Language Basics

- Automatic widening
- Type casting
- Declaring constants

I/O

- Scanners
  - next(), nextInt(), nextLine(), etc.
- PrintStreams (i.e. `System.out`)
  - println(), print(), printf()

Strings

- String literals
- String concatenation (+)
- String comparison
  - compareTo()
  - equals()
  - equalsIgnoreCase()

Arrays

- One dimensional arrays
- Multidimensional arrays
- length field
- Declaring hard-coded arrays

Comments and Whitespace

- Line comment (//)
- Block comment (/*   */)
- Indent properly! Code is unreadable otherwise!

Whitespace Example

```
import java.util.*;
public class x {
public static void main(String[] args) {
Scanner a=new Scanner(System.in);
System.out.print("input");
long c=1;
for(int b=a.nextInt(),d=1;d<=b;++d)c*=d;
System.out.println(c);
} }
```

Whitespace Example

```
public class Factorial {
  public static void main(String[] args) {

       Scanner stdin = new Scanner(System.in);

       System.out.print("Please input a number> ");
       int number = stdin.nextInt();

       long fact = 1;
       for (int i=2; i<=number; ++i) {
               fact*=i;
       }

       System.out.printf("%d! = %d%n",number,fact);

  }
}
```

Errors

- Compilation Errors
- Runtime Errors
- Logic Errors

Class Basics

- Instance Variables
- Instance Methods
- Static Variables
- Static Methods
- Constructors

Terms to know

Instance vs. Static

Constructor

private vs. public

Parameter

Local variable

Declaration

Initialization

Overriding

Overloading

Encapsulation

Information Hiding

Garbage Collection

Object-oriented programming

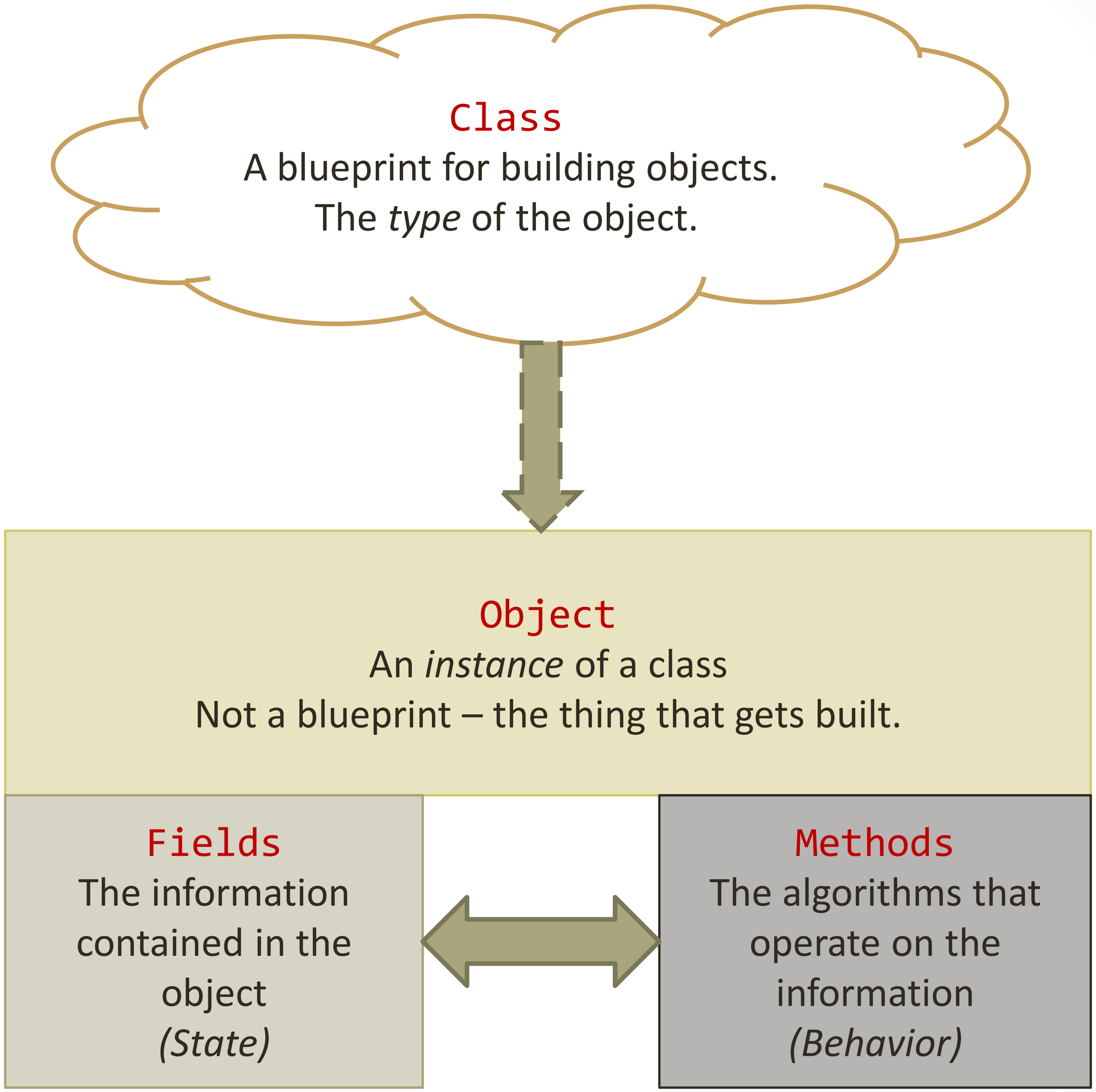
- Ultimately, every program has two elements.
  - Data: The information to process.
  - Functionality: The operations applied to the data.

- The OOP way is to *encapsulate* data and related functionality into an *object*.

- This allows us to program by building abstractions that mimic elements of the problem space.

Objects are abstractions

- A class represents an abstraction that fits the problem.
- Manipulate objects to solve the problem.
- Examples:
  - `String`: a piece of text
  - `InputStream`: A pipe through which data is read in
  - `OutputStream`: A pipe through which data is written out



`Class`
A blueprint for building objects.
The *type* of the object.

`Object`
An *instance* of a class
Not a blueprint – the thing that gets built.

`Fields`
The information contained in the object
(State)

`Methods`
The algorithms that operate on the information
(Behavior)

Fields

- Variables (objects or primitives) contained *inside* an object.
- Their *values* collectively constitute the state of the object.
- Instance variables:
  - Each object of a class has its own copy of instance variables.
- Static variables:
  - Each object of a class shares a single copy of every static variable.
  - Use the `static` modifier to declare them.

Methods

- A method is laid out as follows:

```
modifiers return-type identifier(formal parameters)
{
    body
}
```

- Non-void methods must have a return statement

Methods

- Functions that live within an object.

- Instance methods:
  - Have unrestricted access to all members of the object.
  - Call with `objname.methodName(params…)`
  - `String subword = word.substring(2, 5);`

- Static methods
  - Have unrestricted access to only *static* members of the object.
  - Call with `Classname.methodName(params…)`
  - `double d = Math.random();`

Access modifiers

- There are four modifiers that control who can access class members (i.e., fields and methods)
  - `public`
  - `private`
  - `protected`
  - default (no modifier written)

- `protected` modifier is not important yet.

Access modifiers

- Client of a class: Anything that uses the class.
  - Either by making an object or static members.

- The modifiers affect which fields and methods can be accessed by *clients* of the class.
  - Members with `public` visibility can be directly accessed with the dot operator.
  - Members with `private` visibility cannot be directly accessed by clients.
  - Members with default visibility are public to other classes in the same package, and private to everyone else.

Method overloading

- Can have multiple methods with the same name.
- Method *signatures* have to be different.
- (i.e. two/more methods can have the same name, but only if the parameters are different)

- Signature consists of:
  - Method name
  - Method parameters (types of parameters, and their order)

- Does NOT consist of:
  - Return type
  - Access modifiers

Method overloading

- What will happen?
- public int sum(int a, int b)
- public double sum(int a, int b)

- sum(2,3)
- What will be the result?

Constructors

- Special methods that build the object.
  - Basically initializes the fields.
  - Called as `new Classname(params…);`

- Always has the same name as the class, but no return type.

- Can have zero or more parameters.
- Can be overloaded.

- Cannot use the return statement

The this keyword

- The keyword `this` refers to the current object.
  - From the ‘inside view’, `this` is a reference to the object we’re inside of.
- It is accessible in instance methods and constructors.
- It is NOT accessible in static methods.
  - Makes no real sense in a static context.
- We can use the dot operator with it, regardless of public/private modifiers on class members.
  - Remember – those are only important for the outside view.

Arrays

- Full-fledged objects.

- They contain a public `length` **field** that indicates the size of the array.
  - Contrast with the public `length()` ***method*** of String!

- The `[]` operator can be used to access elements of the array.

Miscellaneous

- The final keyword
- Literals
- Using String
- Using Scanner and PrintStream

- Loops, conditionals, basic operators, etc.

Sample Questions

- True or False
  - “This is not a string literal” is an example of a String literal
  - The following code snippet will compile:

    -
      ```
      int i = 5;
       byte b = (byte)2;
       i = i + b;
      ```
  - Math.random() is an instance method on the Math class
  - Java is the best programming language

Sample Questions

What gets printed out by the following code fragment?

```
int x = 1, y = 2, z = 3;
while (x <= 20) {
     int w = 0;
     for (int i = 0; i < z; ++i)
          w += i;
      x = x + y + z;
      y--;
      z++;
      System.out.println("x = "+x+" y = "+y+" z = "+z);
}
Answers:
```

- x = 6 y = 1 z = 4
- x = 11 y = 0 z = 5
- x = 16 y = -1 z = 6
- x = 21 y = -2 z = 7

Sample Questions

Consider the code fragment below:

```
int x = 5;
while (x) {
     if (x > 10) then
          System.out.println(x);
          x = x/2;
     else
          System.out.println(x);
          x - 1 = x;
 }
```

a) Find four errors in the code fragment that will cause compilation problems.

b) Suggest how to fix these four errors.

c) Using your fixes, what output is given by the code fragment?

Sample Questions

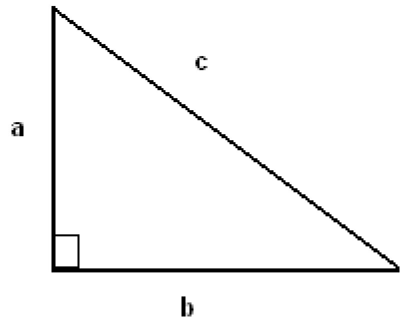
Write a program which will print array elements (int) in reverse order.

Assume array has 10 elements. You need take them from user.

Sample Questions

- Write a program that collects an integer from the user and prints a right triangle of '*'s to the screen with height equal to the number entered by the user

- *
- **
- ***
- ****
- *****
- ******



Sample Questions

- Write a method that takes in a String of all lowercase characters and returns an array of the frequency of each character from ‘a’ – ‘z’ in the indices 0 – 25, respectively.