

# Java Tutorial

Level 0

31-Dec-13

1

## Different Programming Paradigms

- Functional/procedural programming:
  - program is a list of instructions to the computer
- Object-oriented programming
  - program is composed of a collection *objects that communicate with each other*

31-Dec-13

2

## Main Concepts

- Object
- Class
- Inheritance
- Encapsulation

31-Dec-13

3

## Objects

- identity – unique identification of an object
- attributes – data/state
- services – methods/operations
  - supported by the object
  - within objects responsibility to provide these services to other clients

31-Dec-13

4

## Class

- “type”
- object is an **instance** of class
- class groups similar objects
  - same (structure of) attributes
  - same services
- object holds values of its class’s attributes

31-Dec-13

5

## Inheritance

- Class hierarchy
- Generalization and Specialization
  - subclass inherits attributes and services from its superclass
  - subclass may add new attributes and services
  - subclass may reuse the code in the superclass
  - subclasses provide specialized behaviors (overriding and dynamic binding)
  - partially define and implement common behaviors (abstract)

31-Dec-13

6

## Encapsulation

- Separation between internal state of the object and its external aspects
- How ?
  - control access to members of the class
  - **interface** “type”

## What does it buy us ?

- Modularity
  - source code for an object can be written and maintained independently of the source code for other objects
  - easier maintainance and reuse
- Information hiding
  - other objects can ignore implementation details
  - security (object has control over its internal state)
- but
  - shared data need special design patterns (e.g., DB)
  - performance overhead

# JAVA

*mainly for c++ programmer*

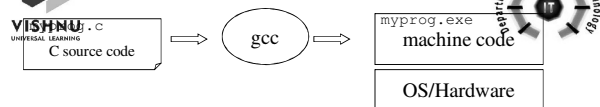
## Why Java ?

- Portable
- Easy to learn
- [ Designed to be used on the Internet ]

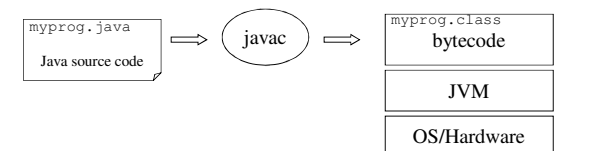
## JVM



- JVM stands for  
**Java Virtual Machine**
- Unlike other languages, Java “executables” are executed on a CPU that does not exist.

### Platform Dependent



### Platform Independent





## Primitive types

- int 4 bytes
- short 2 bytes
- long 8 bytes
- byte 1 byte
- float 4 bytes
- double 8 bytes
- char Unicode encoding (2 bytes)
- boolean {true,false}

*Behaviors is exactly as in C++*

Note:  
Primitive type always begin with lower-case

31-Dec-13






## Primitive types - cont.

- **Constants**
  - 37 integer
  - 37.2 float
  - 42F float
  - 0754 integer (octal)
  - 0xfe integer (hexadecimal)

31-Dec-13

14

## Wrappers



Java provides Objects which wrap primitive types and supply methods.

Example:

```
Integer n = new Integer("4");
int m = n.intValue();
```

31-Dec-13

[Read more about Integer in JDK Documentation](#)

## Hello World

Hello.java



```
class Hello {
    public static void main(String[] args) {
        System.out.println("Hello World !!!");
    }
}
```

C:\javac Hello.java ( compilation creates Hello.class )

C:\java Hello (Execution on the local JVM)

31-Dec-13

16


## More sophisticated

Default C'tor  
Copy C'tor

```
class Kyle {
    private boolean kennyIsAlive_;
    public Kyle() { kennyIsAlive_ = true; }
    public Kyle(Kyle aKyle) {
        kennyIsAlive_ = aKyle.kennyIsAlive_;
    }
    public String theyKilledKenny() {
        if (kennyIsAlive_) {
            kennyIsAlive_ = false;
            return "You bastards !!!";
        } else {
            return "?";
        }
    }
    public static void main(String[] args) {
        Kyle k = new Kyle();
        String s = k.theyKilledKenny();
        System.out.println("Kyle: " + s);
    }
}
```

31-Dec-13

17





## Results


```
javac Kyle.java ( to compile )
java Kyle ( to execute )
Kyle: You bastards !!!
```

31-Dec-13

18



## Arrays



- Array is an object
- Array size is fixed


```
Animal[] arr; // nothing yet ...

arr = new Animal[4]; // only array of pointers


for(int i=0 ; i < arr.length ; i++) {
    arr[i] = new Animal();
}

// now we have a complete array
```

31-Dec-13
19



## Arrays - Multidimensional



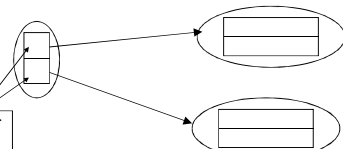
- In C++
 

```
Animal arr[2][2]
```

Is: 


- In Java
 

```
Animal [][] arr=
new Animal[2][2]
```




What is the type of the object here ?

31-Dec-13
20



## Static - [1/4]



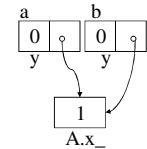
**Member data** - Same data is used for all the instances (objects) of some Class.

```
Class A {
    public int y = 0;
    public static int x_ = 1;
};


A a = new A();
A b = new A();
System.out.println(b.x_);
a.x_ = 5;
System.out.println(b.x_);
A.x_ = 10;
System.out.println(b.x_);
```

*Assignment performed on the first access to the Class. Only one instance of 'x' exists in memory*


**Output :**  
1  
5  
10



31-Dec-13
21



## Static - [2/4]




**Member function**

- Static member function can access only static members
- Static member function can be called without an instance.


```
Class TeaPot {
    private static int numOfTP = 0;
    private Color myColor_;
    public TeaPot(Color c) {
        myColor_ = c;
        numOfTP++;
    }
    public static int howManyTeaPots() {
        return numOfTP;
    }

    // error :
    public static Color getColor() {
        return myColor_;
    }
}
```

31-Dec-13
22



## Static - [2/4] cont.




**Usage :**


```
TeaPot tp1 = new TeaPot(Color.RED);
TeaPot tp2 = new TeaPot(Color.GREEN);

System.out.println("We have " +
    TeaPot.howManyTeaPots() + " Tea Pots");
```

31-Dec-13
23



## Static - [3/4]



**Block**

- Code that is executed in the first reference to the class.
- Several static blocks can exist in the same class ( Execution order is by the appearance order in the class definition ).
- Only static members can be accessed.

```
class RandomGenerator {
    private static int seed;

    static {
        int t = System.getTime() % 100;
        seed = System.getTime();
        while(t-- > 0)
            seed = getNextNumber(seed);
    }
}
```

31-Dec-13
24

## String is an Object

- Constant strings as in C, does not exist
- The function call `foo("Hello")` creates a String object, containing "Hello", and passes reference to it to `foo`.
- There is no point in writing :  
  

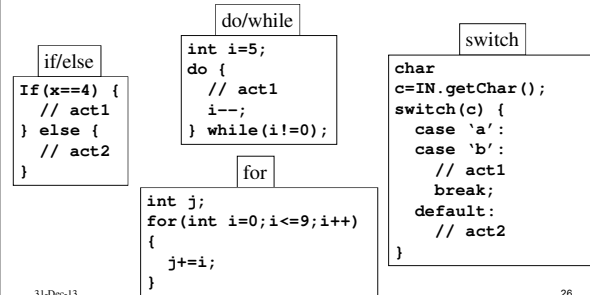
```
String s = new String("Hello");
```
- The String object is a constant. It can't be changed using a reference to it.

31-Dec-13

25

## Flow control

Basically, it is exactly like c/c++.



31-Dec-13

26

## Packages

- Java code has hierarchical structure.
- The environment variable CLASSPATH contains the directory names of the roots.
- Every Object belongs to a package ( 'package' keyword)
- Object full name contains the name full name of the package containing it.

31-Dec-13

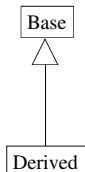
27

## Access Control

- **public** member (function/data)
  - Can be called/modified from outside.
- **protected**
  - Can be called/modified from derived classes
- **private**
  - Can be called/modified only from the current class
- **default ( if no access modifier stated )**
  - Usually referred to as "Friendly".
  - Can be called/modified/instantiated from the same package.

31-Dec-13

28



```

class Base {
    Base() {}
    Base(int i) {}
    protected void foo() {...}
}

class Derived extends Base {
    Derived() {}
    protected void foo() {...}
    Derived(int i) {
        super(i);
        ...
        super.foo();
    }
}
    
```

As opposed to C++, it is possible to inherit only from ONE class.

**Pros** avoids many potential problems and bugs.

**Cons** might cause code replication

31-Dec-13

29

## Polymorphism

- Inheritance creates an "is a" relation:


For example, if B inherits from A, then we say that "B is also an A".

Implications are:

- access rights (Java forbids reducing access rights) - derived class can receive all the messages that the base class can.
- behavior
- precondition and postcondition

31-Dec-13

30



## Inheritance (2)


In Java, all methods are virtual :

```

class Base {
    void foo() {
        System.out.println("Base");
    }
}
class Derived extends Base {
    void foo() {
        System.out.println("Derived");
    }
}
public class Test {
    public static void main(String[] args) {
        Base b = new Derived();
        b.foo(); // Derived.foo() will be activated
    }
}

```

31-Dec-13 31




## Inheritance (3) - Optional

```

classC extends classB {
    classC(int arg1, int arg2){
        this(arg1);
        System.out.println("In classC(int arg1, int arg2)");
    }
    classC(int arg1){
        super(arg1);
        System.out.println("In classC(int arg1)");
    }
}
class classB extends classA {
    classB(int arg1){
        super(arg1);
        System.out.println("In classB(int arg1)");
    }
    classB(){
        System.out.println("In classB()");
    }
}

```

31-Dec-13 32




## Inheritance (3) - Optional

```

classA {
    classA(int arg1){
        System.out.println("In classA(int arg1)");
    }
    classA(){
        System.out.println("In classA()");
    }
}
class classB extends classA {
    classB(int arg1, int arg2){
        this(arg1);
        System.out.println("In classB(int arg1, int arg2)");
    }
    classB(int arg1){
        super(arg1);
        System.out.println("In classB(int arg1)");
    }
}
class B() {
    System.out.println("In classB()");
}

```

31-Dec-13 33



## Abstract

Abstract member function, means that the function does not have an implementation.

- abstract** class, is class that can not be instantiated.


```

AbstractTest.java:6: class AbstractTest is an abstract class.
it can't be instantiated.
    new AbstractTest();
    ^
1 error

```

**NOTE:**  
An abstract class is not required to have an abstract method in it. But any class that has an abstract method in it or that does not provide an implementation for any abstract methods declared in its superclasses must be declared as an abstract class.

31-Dec-13 **Example** →



## Abstract - Example


```

package java.lang;
public abstract class Shape {
    public abstract void draw();
    public void move(int x, int y) {
        setColor(BackGroundColor);
        draw();
        setCenter(x,y);
        setColor(ForeGroundColor);
        draw();
    }
}

package java.lang;
public class Circle extends Shape {
    public void draw() {
        // draw the circle ...
    }
}

```

31-Dec-13 35




## Interface

Interfaces are useful for the following:


- Capturing similarities among unrelated classes without artificially forcing a class relationship.
- Declaring methods that one or more classes are expected to implement.
- Revealing an object's programming interface without revealing its class.

31-Dec-13 36



# Interface


## abstract "class"




- Helps defining a "usage contract" between classes
- All methods are public
- Java's compensation for removing the multiple inheritance. You can "inherit" as many interfaces as you want.

\* - The correct term is "to implement" an interface


### Example



37



# Interface



```

interface IChef {
    void cook(Food food);
}

interface BabyKicker {
    void kickTheBaby(Baby);
}


interface SouthParkCharacter {
    void curse();
}

class Chef implements IChef, SouthParkCharacter {
    // overridden methods MUST be public
    // can you tell why ?
    public void curse() { ... }
    public void cook(Food f) { ... }
}
    
```


\* access rights (Java forbids reducing of access rights)

31-Dec-13

38




# When to use an interface ?




Perfect tool for encapsulating the classes inner structure. Only the interface will be exposed

31-Dec-13

39




# Collections




- Collection/container
  - object that groups multiple elements
  - used to store, retrieve, manipulate, communicate aggregate data
- Iterator - object used for traversing a collection and selectively remove elements
- Generics – implementation is parametric in the type of elements

31-Dec-13

40



# Java Collection Framework




- Goal: Implement reusable data-structures and functionality
- Collection interfaces - manipulate collections independently of representation details
- Collection implementations - reusable data structures
 


```
List<String> list = new ArrayList<String>(c);
```
- Algorithms - reusable functionality
  - computations on objects that implement collection interfaces
  - e.g., searching, sorting
  - polymorphic: the same method can be used on many different implementations of the appropriate collection interface

31-Dec-13

41



# Collection Interfaces





```

graph TD
    Collection --> Set
    Collection --> List
    Collection --> Queue
    Set --> SortedSet
    Map --> SortedMap
    
```

31-Dec-13

42



## Collection Interface

### Basic Operations

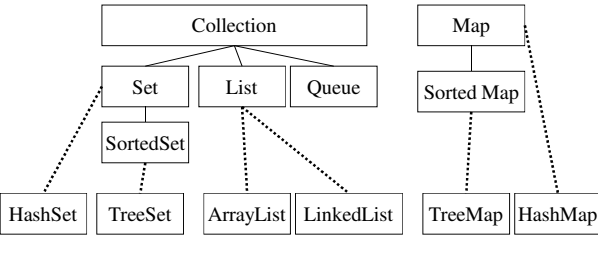
- int size();
- boolean isEmpty();
- boolean contains(Object element);
- boolean add(E element);
- boolean remove(Object element);
- Iterator iterator();

- Bulk Operations**
  - boolean containsAll(Collection<?> c);
  - boolean addAll(Collection<? extends E> c);
  - boolean removeAll(Collection<?> c);
  - boolean retainAll(Collection<?> c);
  - void clear();
- Array Operations**
  - Object[] toArray();
  - <T> T[] toArray(T[] a);

31-Dec-13
43

## General Purpose Implementations





```

List<String> list1 = new ArrayList<String>(c);
List<String> list2 = new LinkedList<String>(c);

```

31-Dec-13
44

## final

- final member data**  
Constant member
- final member function**  
The method can't be overridden.
- final class**  
'Base' is final, thus it can't be extended

```



final class Base {
    final int i=5;
    final void foo() {
        i=10;
        //what will the compiler say about this?
    }
}

class Derived extends Base {
    // Error
    // another foo ...
    void foo() {
    }
}

```

(String class is final)

31-Dec-13
45

## final

```

// java:6: Can't subclass final classes: class Base
class class Derived extends Base {
    ^
1 error
}

```



```

final class Base {
    final int i=5;
    final void foo() {
        i=10;
    }
}

class Derived extends Base {
    // Error
    // another foo ...
    void foo() {
    }
}

```

31-Dec-13
46

## IO - Introduction

### Definition

- Stream** is a flow of data
  - characters read from a file
  - bytes written to the network
  - ...



### Philosophy

- All streams in the world are basically the same.
- Streams can be divided (as the name "IO" suggests) to **Input** and **Output** streams.

### Implementation

- Incoming flow of data (characters) implements "Reader" (InputStream for bytes)
- Outgoing flow of data (characters) implements "Writer" (OutputStream for bytes -eg. Images, sounds etc.)

31-Dec-13
47

## Exception - What is it and why do I care?

**Definition:** An *exception* is an event that occurs during the execution of a program that disrupts the normal flow of instructions.

- Exception is an Object
- Exception class must be descendent of Throwable.

31-Dec-13
48



## Exception - What is it and why do I care (2)

By using exceptions to manage errors, Java programs have the following advantages over traditional error management techniques:

- 1: Separating Error Handling Code from "Regular" Code
- 2: Propagating Errors Up the Call Stack
- 3: Grouping Error Types and Error Differentiation

31-Dec-13

49

## 1: Separating Error Handling Code from "Regular" Code (1)

```
readFile {
    open the file;
    determine its size;
    allocate that much memory;
    read the file into memory;
    close the file;
}
```

31-Dec-13

50

## 1: Separating Error Handling Code from "Regular" Code (2)

```
enum ErrorCodeType {
    ...
}

readFile {
    initialize errorCode = 0;
    open the file;
    if (theFileIsOpen) {
        determine the length of the file;
        if (gotTheFileLength) {
            allocate that much memory;
            if (gotEnoughMemory) {
                read the file into memory;
                if (readFailed) {
                    errorCode = -1;
                }
            } else {
                errorCode = -2;
            }
        } else {
            errorCode = -3;
        }
    }
    close the file;
    if (theFileDintClose && errorCode == 0) {
        errorCode = -4;
    } else {
        errorCode = errorCode and -4;
    }
    else {
        errorCode = -5;
    }
}

return errorCode;
```

31-Dec-13

51

## 1: Separating Error Handling Code from "Regular" Code (3)

```
readFile {
    try {
        open the file;
        determine its size;
        allocate that much memory;
        read the file into memory;
        close the file;
    } catch (fileOpenFailed) {
        doSomething;
    } catch (sizeDeterminationFailed) {
        doSomething;
    } catch (memoryAllocationFailed) {
        doSomething;
    } catch (readFailed) {
        doSomething;
    } catch (fileCloseFailed) {
        doSomething;
    }
}
```

31-Dec-13

52

## 2: Propagating Errors Up the Call Stack



```
method1 {
    try {
        call method2;
    } catch (exception) {
        doErrorProcessing;
    }
}

method2 throws exception {
    call method3;
}

method3 throws exception {
    call readFile;
}
```



31-Dec-13

53



## JAVA BASICS

### Level 1





### Comments are almost like C++

The javadoc program generates HTML API documentation from the "javadoc" style comments in your code.

```
/* This kind of comment can span multiple lines */
// This kind is to the end of the line
/**
 * This kind of comment is a special
 * 'javadoc' style comment
 */
```

31-Dec-13 2





### An example of a class

```
class Person {
    String name;
    int age;
    void birthday ( ) {
        age++;
        System.out.println (name +
            ' is now ' + age);
    }
}
```

Variable

Method

31-Dec-13 3



### Scoping



- As in C/C++, scope is determined by the placement of curly braces {}.
- A variable defined within a scope is available only to the end of that scope.

```
{ int x = 12;
  /* only x available */
  { int q = 96;
    /* both x and q available */
  }
  /* only x available */
  /* q "out of scope" */
}
```

This is ok in C/C++ but not in Java.

```
{ int x = 12;
  { int x = 96; /* illegal */
  }
}
```



31-Dec-13 4



### An array is an object

- Person mary = new Person ( );
- int myArray[ ] = new int[5];
- int myArray[ ] = {1, 4, 9, 16, 25};
- String languages [ ] = {"Prolog", "Java"};
- Since arrays are objects they are allocated dynamically
- Arrays, like all objects, are subject to garbage collection when no more references remain
  - so fewer memory leaks
  - Java doesn't have pointers!

31-Dec-13 5





### Scope of Objects

- Java objects don't have the same lifetimes as primitives.
- When you create a Java object using **new**, it hangs around past the end of the scope.
- Here, the scope of name **s** is delimited by the {}s but the String object hangs around until GC'd

```
{
    String s = new String("a
    string");
} /* end of scope */
```

31-Dec-13 6

# CMSC 331 Principles of programming languages



## Methods, arguments and return values



- Java methods are like C/C++ functions. General case:

```
returnType methodName ( arg1, arg2, ... argN) {  
    methodBody  
}
```

The return keyword exits a method optionally with a value

```
int storage(String s) {return s.length() * 2;}  
boolean flag() { return true; }  
float naturalLogBase() { return 2.718f; }  
void nothing() { return; }  
void nothing2() {}
```

31-Dec-13 7





## The static keyword

- Java methods and variables can be declared static
- These exist **independent of any object**
- This means that a Class's
  - static methods can be called even if no objects of that class have been created and
  - static data is "shared" by all instances (i.e., one rvalue per class instead of one per instance

```
class StaticTest {static int i = 47;}  
StaticTest st1 = new StaticTest();  
StaticTest st2 = new StaticTest();  
// st1.i == st2.i == 47  
StaticTest.i++; // or st1.i++ or st2.i++  
// st1.i == st2.i == 48
```



31-Dec-13 8





## Array Operations

- Subscripts always start at 0 as in C
- Subscript checking is done automatically
- Certain operations are defined on arrays of objects, as for other classes
  - e.g. myArray.length == 5

31-Dec-13 9





# Example Programs



## Echo.java

```
C:\UMBC\331\java>type echo.java  
// This is the Echo example from the Sun tutorial  
class echo {  
    public static void main(String args[]) {  
        for (int i=0; i < args.length; i++) {  
            System.out.println( args[i] );  
        }  
    }  
}  
  
C:\UMBC\331\java>javac echo.java  
  
C:\UMBC\331\java>java echo this is pretty silly  
this  
is  
pretty  
silly  
  
C:\UMBC\331\java>
```

31-Dec-13 11





## Factorial Example

From Java

```
/* This program computes the factorial of a number */  
public class Factorial { // Define a class  
    public static void main(String[] args) { // The program starts here  
        int input = Integer.parseInt(args[0]); // Get the user's input  
        double result = factorial(input); // Compute the factorial  
        System.out.println(result); // Print out the result  
    } // The main() method ends here  
  
    public static double factorial(int x) { // This method computes x!  
        if (x < 0) // Check for bad input  
            return 0.0; // if bad, return 0  
        double fact = 1.0; // Begin with an initial value  
        while(x > 1) { // Loop until x equals 1  
            fact = fact * x; // multiply by x each time  
            x = x - 1; // and then decrement x  
        } // Jump back to the start of loop  
        return fact; // Return the result  
    } // factorial() ends here  
} // The class ends here
```

31-Dec-13 12



# CMSC 331 Principles of programming languages



## JAVA Classes

- The *class* is the fundamental concept in JAVA (and other OOPs)
- A class describes some data object(s), and the operations (or methods) that can be applied to those objects
- Every object and method in Java belongs to a class
- Classes have data (fields) and code (methods) and classes (member classes or inner classes)
- Static methods and fields belong to the class itself
- Others belong to instances

31-Dec-13 13



## Example



```
public class Circle {
    // A class field
    public static final double PI= 3.14159;    // A useful constant

    // A class method: just compute a value based on the arguments
    public static double radiansToDegrees(double rads) {
        return rads * 180 / PI;
    }

    // An instance field
    public double r;    // The radius of the circle

    // Two methods which operate on the instance fields of an object
    public double area() {    // Compute the area of the circle
        return PI * r * r;
    }
    public double circumference() {    // Compute the circumference of the circle
        return 2 * PI * r;
    }
}
```



31-Dec-13 14



## Constructors

- Classes should define one or more methods to create or construct instances of the class
- Their name is the same as the class name
  - note deviation from convention that methods begin with lower case
- Constructors are differentiated by the number and types of their arguments
  - An example of overloading
- If you don't define a constructor, a default one will be created.
- Constructors automatically invoke the zero argument constructor of their superclass when they begin (note that this yields a recursive process!)

31-Dec-13 15



## Constructor example

```
public class Circle {
    public static final double PI = 3.14159;    // A constant
    public double r;    // instance field holds circle's radius

    // The constructor method: initialize the radius field
    public Circle(double r) { this.r = r; }



    // Constructor to use if no arguments
    public Circle() { r = 1.0; }
    // better: public Circle() { this(1.0); }

    // The instance methods: compute values based on radius
    public double circumference() { return 2 * PI * r; }
    public double area() { return PI * r*r; }
}
```

*this.r refers to the r field of the class*

*This() refers to a constructor for the class*

31-Dec-13 16





## Extending a class

Class hierarchies reflect subclass-superclass relations among classes.

- One arranges classes in hierarchies:
  - A class inherits instance variables and instance methods from all of its superclasses. Tree -> BinaryTree -> BST
  - You can specify only ONE superclass for any class.
- When a subclass-superclass chain contains multiple instance methods with the same signature (name, arity, and argument types), the one **closest** to the target instance in the subclass-superclass chain is the one executed.
  - All others are shadowed/overridden.
- Something like multiple inheritance can be done via interfaces (more on this later)
- What's the superclass of a class defined without an extends clause?

31-Dec-13 17



## Extending a class


```
class PlaneCircle extends Circle {
    // It automatically inherits the fields and methods of Circle,
    // so we only have to put the new stuff here.
    // New instance fields that store the center point of the circle
    public double cx, cy;

    // A new constructor method to initialize the new fields
    // It uses a special syntax to invoke the Circle() constructor
    public PlaneCircle(double r, double x, double y) {
        super(r);    // Invoke the constructor of the superclass, Circle()
        this.cx = x;    // Initialize the instance field cx
        this.cy = y;    // Initialize the instance field cy
    }

    // The area() and circumference() methods are inherited from Circle
    // A new instance method that checks whether a point is inside the circle
    // Note that it uses the inherited instance field r
    public boolean isInside(double x, double y) {
        double dx = x - cx, dy = y - cy;    // Distance from center
        double distance = Math.sqrt(dx*dx + dy*dy);    // Pythagorean theorem
        return (distance < r);
    }
}
```

31-Dec-13 18

# CMSC 331 Principles of programming languages




VISHNU  
UNIVERSAL LEARNING

Department of Information Technology

## Overloading, overwriting, and shadowing

- **Overloading** occurs when Java can distinguish two procedures with the same name by examining the number or types of their parameters.
- **Shadowing** or **overriding** occurs when two procedures with the same signature (name, the same number of parameters, and the same parameter types) are defined in different classes, one of which is a superclass of the other.

31-Dec-1319




VISHNU  
UNIVERSAL LEARNING

Department of Information Technology

## On designing class hierarchies

- Programs should obey the *explicit-representation principle*, with classes included to reflect natural categories.
- Programs should obey the *no-duplication principle*, with instance methods situated among class definitions to facilitate sharing.
- Programs should obey the *look-it-up principle*, with class definitions including instance variables for stable, frequently requested information.
- Programs should obey the *need-to-know principle*, with public interfaces designed to restrict instance-variable and instance-method access, thus facilitating the improvement and maintenance of nonpublic program elements.
- If you find yourself using the phrase *an X is a Y* when describing the relation between two classes, then the X class is a subclass of the Y class.
- If you find yourself using *X has a Y* when describing the relation between two classes, then instances of the Y class appear as parts of instances of the X class.

31-Dec-1320




VISHNU  
UNIVERSAL LEARNING

Department of Information Technology

## Data hiding and encapsulation

- Data-hiding or encapsulation is an important part of the OO paradigm.
- Classes should carefully control access to their data and methods in order to
  - Hide the irrelevant implementation-level details so they can be easily changed
  - Protect the class against accidental or malicious damage.
  - Keep the externally visible class simple and easy to document
- Java has a simple access control mechanism to help with encapsulation

31-Dec-1321



VISHNU  
UNIVERSAL LEARNING

Department of Information Technology

## Example

```
package shapes;           // Specify a package for the class
public class Circle {     // The class is still public
    public static final double PI = 3.14159;

    protected double r;   // Radius is hidden, but visible to subclasses


    // A method to enforce the restriction on the radius
    // This is an implementation detail that may be of interest to subclasses
    protected checkRadius(double radius) {
        if (radius < 0.0)
            throw new IllegalArgumentException("radius may not be negative.");
    }

    // The constructor method
    public Circle(double r) { checkRadius(r); this.r = r; }

    // Public data accessor methods
    public double getRadius() { return r; }
    public void setRadius(double r) { checkRadius(r); this.r = r; }

    // Methods to operate on the instance field
    public double area() { return PI * r * r; }
    public double circumference() { return 2 * PI * r; }
}
```

31-Dec-1322




VISHNU  
UNIVERSAL LEARNING

Department of Information Technology

## Access control

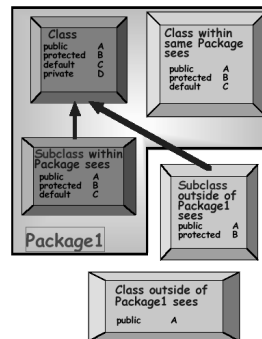
- Access to packages
  - Java offers no control mechanisms for packages.
  - If you can find and read the package you can access it
- Access to classes
  - All top level classes in package P are accessible anywhere in P
  - All public top-level classes in P are accessible anywhere
- Access to class members (in class C in package P)

31-Dec-1323



VISHNU  
UNIVERSAL LEARNING



Department of Information Technology



A summary of Java scoping visibility

31-Dec-1324



# CMSC 331 Principles of programming languages



## Getters and setters

- A getter is a method that extracts information from an instance.
  - One benefit: you can include additional computation in a getter.
- A setter is a method that inserts information into an instance (also known as mutators).
  - A setter method can check the validity of the new value (e.g., between 1 and 7) or trigger a side effect (e.g., update a display)
- Getters and setters can be used even without underlying matching variables
- Considered good OO practice
- Essential to java beans
- Convention: for variable fooBar of type fbtype, define
  - getFooBar()
  - setFooBar(fbtype x)

31-Dec-13 25



```
package shapes; // Specify a package for the class

public class Circle { // The class is still public
    // PI is a generally useful constant, so we keep it public
    static final double PI = 3.14159;

    protected double r; // Radius is hidden, but visible to subclasses



    // A method to enforce the restriction on the radius
    // This is an implementation detail that may be of interest to subclasses
    protected checkRadius(double radius) {
        if (radius < 0.0)
            throw new IllegalArgumentException("radius may not be negative.");
    }

    // The constructor method
    public Circle(double r) { checkRadius(r); this.r = r; }

    // Public data accessor methods
    public double getRadius() { return r; }
    public void setRadius(double r) { checkRadius(r); this.r = r; }

    // Methods to operate on the instance field
    public double area() { return PI * r * r; }
    public double circumference() { return 2 * PI * r; }
}
```



31-Dec-13 26



## Abstract classes and methods

- Abstract vs. concrete classes
- Abstract classes can not be instantiated
- An abstract method is a method w/o a body
- (Only) Abstract classes can have abstract methods
- In fact, any class with an abstract method is automatically an abstract class

31-Dec-13 27





```
public abstract class Shape {
    public abstract double area(); // Abstract methods: note
    public abstract double circumference(); // semicolon instead of body
}

class Circle extends Shape {
    public static final double PI = 3.14159265358979323846;
    protected double r; // Instance data
    public Circle(double r) { this.r = r; } // Constructor
    public double getRadius() { return r; } // Accessor
    public double area() { return PI*r*r; } // Implementations of
    public double circumference() { return 2*PI*r; } // abstract methods.
}

class Rectangle extends Shape {
    protected double w, h; // Instance data
    public Rectangle(double w, double h) { // Constructor
        this.w = w; this.h = h;
    }
    public double getWidth() { return w; } // Accessor method
    public double getHeight() { return h; } // Another accessor
    public double area() { return w*h; } // Implementations of
    public double circumference() { return 2*(w + h); } // abstract methods.
}
```



31-Dec-13 28



## Syntax Notes



- No global variables
  - class variables and methods may be applied to any instance of an object
  - methods may have local (private?) variables
- No pointers
  - but complex data objects are “referenced”
- Other parts of Java are borrowed from PL/I, Modula, and other languages

31-Dec-13 29

A little cup of Java-coffee  
Level 2



31-Dec-13 1

## Today's session

- Part-1) Java overview (5mins)
  - What java is
  - Java features
  - Java's cross-platform
- Part-2) two simple and typical java programs
  - A stand-alone java and its running (5mins)
  - A applet and its running (5mins)
- Part-3) how to learn java by yourself (5mins)
  - 3 stages
  - resources



31-Dec-13 2

## Part-one

- Java overview



31-Dec-13 3

## What Java is

- Java is an "easy" programming language.
  - just a tool like C++, VB, ...and English. Somehow a language tool itself is not so complex.
- Java works for internet project(mainly), and apply "3-tired architecture", coding on the server-side
  - So besides Java language knowledge, we need to learn lots of thing about telecommunication on WEB, to finish a real-time project.



31-Dec-13 4

## What Java is(continue)

- Java applies Object-Oriented Tech.
  - Java is not so difficulty, though OOP is. A java expert must be an OOP expert.
- Java is slower than C++ ( 3-5 times), Java's database function is slower than VB.
- Java is very portable: cross-platform

31-Dec-13 5

## Java's Features



- Simple
 

Java omits many rarely used, poorly understood, confusing features of C++. Say : No Pointer! No dynamic delete.
- Object Oriented
 

Object –oriented design is a technology that focuses design on the data (object) and on the interfaces to it.

Let's say, everything is an object, everything will become a class in Java. Every java program, in top- level view, is classes.

31-Dec-13 6

## Java's Features(continue)

- Distributed
 



Basically, Java is for Net-Work application, for WEB project.

Java can open and access "objects" across the Net via URLs (Uniform Resource Locator)----eg.  
 "http://gamut.neiu.edu/~ylei/home.html",

with the same ease as when accessing a local file system

31-Dec-13

7



## Java's Features(continue)

- Robust
 

The single biggest difference between Java and C/C++ is that Java has "a inner safe pointer-model", therefore it eliminates the possibility of overwriting memory and corrupting data, so programmers feel very safe in coding.

31-Dec-13

8



## Java's Features(continue)

- GUI [Java-Swing]
 

For some reason, Sun believe their java-swing is very important, so they always put it in their certificate-tests.
- Multi-threaded
- Secure [ Exception handling ]
- Dynamic [ for Server-side coding]

31-Dec-13

9

## Java's cross-platform

- Interpreted Execute: cross-platform
 

**why:** For cross-platform purpose. **Once coding, run anywhere.**

The **Java interpreter** ( **java.exe** and its **javaVirtualMachine**) can execute compiled Java-byte-codes(**xxx.class**) directly **on any machine** to which the interpreter has been ported.



**How:** ( eg. **Dos command line style**)

  - Edit source code "**demo.java**", by notepad/or other IDE tools
  - Compile ( **javac.exe** ) "**demo.java**"→ **javac Demo.java** → Java byte codes, namely, Demo.class
  - Execute (Interpreted Execute) **java Demo**

**Speed issue AND new solutions:** java is slower than c++ in running. however, by now, there are some new technology of Java compiler, such as "Just-in-time", and "HotSpot adaptive Compiler". They make java very faster than before.

31-Dec-13

10

Ps: Compiler and Interpreters: Run in Physical CPU

1. **Compilers** use the traditional compile/link/run strategy.  
*Examples:* C, C++, ML.

```

source [comple] native-files [link] nativeprogram [run]
demo.c ----> obj      -> demo.exe --> Intel cpu
Demoh.h ----->
  
```



2. **Interpreters** execute the source code directly. *Examples:* BASIC, Perl, TCL/Tk, ML.

```

source [load]                                [interpret run]
demo.perl -> source-program -> -> Intel cpu
data ----->
  
```

31-Dec-13

11

## Java: Run in Virtual Cpu :cross-platfrom

```


Demo.java→ Compile →Demo.class→ link→ xxx.class
Source-code  "javac"  byte-code files      bytecode program
  
```

→interpretedly run on VM |--> Intel CPU  
 (virtual CPU: JSDK ) |--> ... CPU  
 |--> Apple CPU


31-Dec-13

12






## Part-2 2 samples




- How many kinds of java programs ?
- Demo-1: Stand-lone sample
- Demo-2: an Applet sample

31-Dec-13
13




## How many kinds of Java Programs?




- Un-network app.: (1)Standalone Java program (today)
- Network app: non-standalone Java program
  - Internet: (2)Applet , (today)
  - (3)servlet
  - (4)JavaBean classes
  - Intranet: (5)EJB ( EnterpriseJavaBean ),
  - (6)RMI, etc

31-Dec-13
14



## Standalone Java Program





- The main() method  

```
public static void main(String args[]){
    ...
}
```

public--- the interpreter can call it  
static ----It is a static method belonging to the class  
void ----It does not return a value  
String----It always has an array of String objects as its formal parameter.  
the array contains any arguments passed to the program on the command line  
the source file's name must match the class name which main method is in

31-Dec-13
15

```



1 // Fig. 2.1: 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!

Program Output

31-Dec-13
16

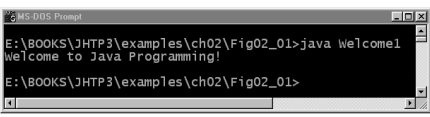



```


1 // Fig. 2.1: 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 }

```


Java program



31-Dec-13
17



## A Simple GUI Program: Printing a Line of Text




- Display
  - Most Java applications use windows or a dialog box
    - We have used command window
  - Class **JOptionPane** allows us to use dialog boxes
- Packages
  - 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
    - Package has classes for using Graphical User Interfaces (GUIs)


31-Dec-13
18

```


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 to Java Programming!" );
10
11         System.exit( 0 ); // terminate the program
12     }
13 }

```






31-Dec-13
19




## Packages




- Like “namespace” in C++
- How to use:
  - C++: using namespace xxx
  - Java: import xxx, or  
import xxx.xx

31-Dec-13
20



## A Simple Java Applet: Drawing a String



```

1 <html>
2 <applet code="WelcomeApplet.class" width=300 height=30>
3 </applet>
4 </html>

```


– **appletviewer** only understands **<applet>** tags

- Ignores everything else
- Minimal browser

– Executing the applet

- appletviewer WelcomeApplet.html**
- Perform in directory containing **.class** file


31-Dec-13
21



```

1 // Fig. 3.6: WelcomeApplet.java
2 // A first applet in Java
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
5
6 public class WelcomeApplet extends JApplet {
7     public void paint( Graphics g )
8     {
9         g.drawString( "Welcome to Java Programming!", 25, 25 );
10     }
11 }


```



**import** allows us to use predefined classes (allowing us to use applets and graphics, in this case).

**extends** allows us to inherit the capabilities of class **JApplet**.

Method **paint** is guaranteed to be called in all applets. Its first line must be defined as above.




31-Dec-13
22


```

1 // Fig. 3.8: WelcomeApplet2.java
2 // Displaying multiple strings
3 import javax.swing.JApplet; // import class JApplet
4 import java.awt.Graphics; // import class Graphics
5
6 public class WelcomeApplet2 extends JApplet {
7     public void paint( Graphics g )
8     {
9         g.drawString( "Welcome to", 25, 25 );
10        g.drawString( "Java Programming!", 25, 40 );
11    }
12 }

```



The two **drawString** statements simulate a newline. In fact, the concept of lines of text does not exist when drawing strings.




31-Dec-13
23


```

1 // Displaying text and lines
2 import javax.swing.JApplet; // import class JApplet
3 import java.awt.Graphics; // import class Graphics
4
5 public class WelcomeLines extends JApplet {
6     public void paint( Graphics g )
7     {
8         g.drawLine( 15, 10, 210, 10 );
9         g.drawLine( 15, 30, 210, 30 );
10        g.drawString( "Welcome to Java Programming!", 25, 25 );
11    }
12 }



```



Draw horizontal lines with **drawLine** (endpoints have same y coordinate).





31-Dec-13
24

## Part-3

- How to learn Java by ourself



31-Dec-13 25

## 3 stages

- S-1: basic
  - Contents: language grammars + GUI (swings and event-driven) Applets
  - 2-4 weeks
- S-2: mid-level projects
  - Contents:
    - Exception Handling
    - Threads
    - Streams
    - Network
  - 4-8 weeks



31-Dec-13 26

## 3 Stages(cont'd)

S-3: Advanced projects  
 contents: JavaBeans  
           RMI  
           Servlets and JSP  
           EJB...  
           many topics  
 time: years , just do projects with Java



31-Dec-13 27

## Self-training Resources: in Stage-1 and Stage-2

- Sun's free JSDK. Download and install it.
  - By the way, many books give us a free CD of JSDK.
  - Visit <http://orion.neiu.edu/~ncaftori/>
- Online books <<Thinking in Java>>, it has many translated version, Japanese, Chinese, etc.
- Sun's web training
- Other books:
  - Sun's <<core java>>, it's the base of Sun's certificate-tests.
  - <<Java:How to program>>, html style, friendly
    - Search in <http://deitel.com>, a lots of sample codes

31-Dec-13 28

## IDE's: search Sun's web: sun.java.com

- Jbuilder
- Visual Age
- Sun Forte
- Visual Café
- J++

31-Dec-13 29

## Java Programming Assignments

### Part – 1

---

**1. Write a program in Java to check if a number is even or odd in Java?** (input 2 output true, input 3 : output false)

A number is called even if it is completely divisible by two and odd if it's not completely divisible by two. For example number 4 is even number because when you do  $4/2$ , remainder is 0 which means 4 is completely divisible by 2. On the other hand 5 is odd number because  $5/2$  will result in remainder as 1

**2. Write a program in Java to find out if a number is prime in Java?** (input 7: output true, input 9 : output false)

A number is called prime if it is divisible by either itself or 1. There are many algorithm to find prime numbers e.g. instead of dividing till number, division upto square root of number may be enough. Start from simplest one and then try to solve this problem with couple of different ways.

**3. Write Java program to check if a number is palindrome in Java?** (121 is palindrome, 321 is not)

A number is called a palindrome if number is equal to reverse of number e.g. 121 is palindrome because reverse of 121 is 121 itself. On the other hand 321 is not palindrome because reverse of 321 is 123 which is not equal to 321

**4. How to find if a number is power of 2 in Java?** (1,2, 4 power of 2, 3 is not)

This is another interesting Java programming exercise. This program can be solved using different ways e.g. using arithmetic operator or by using bit shift operator.

**5. Write program to sort an integer array without using API methods?**

Sorting questions are one of the integral part of programming questions. There are many sorting algorithm out there to sort any array in Java e.g. Bubble sort, Insertion sort, Selection sort or quick sort. Implementing sorting algorithm itself a good programming exercise in Java.

**6. Write Java program to check if a number is Armstrong number or not?** (input 153 output true, 123 output false)

An Armstrong number of 3 digit is a number for which sum of cube of its digits are equal to number e.g. 371 is an Armstrong number because  $3*3*3 + 7*7*7 + 1*1*1 = 371$ )

**7. Write a program in Java to reverse any String without using StringBuffer?**

This is another classical Java programming question. You can reverse String in various way in Java but two programming technique is used to do e.g. Iteration and Recursion. Try solving this problem using Iteration first by using Java's arithmetic operator and then look to implement a recursive solution.

**8. Write a program in Java to print Fibonacci series up to given number? Write both iterative and recursive version.**

Fibonacci series a popular number series and very popular programming question in Java, in which number is equal to sum of previous two numbers, starting from third. Fibonacci series is also a good recursion exercise and often asked in Interviews as well. Try doing this exercise by using both Iteration e.g. loops and recursion.

## Java Programming Assignments

**9. Write a Java program to calculate factorial of an integer number ? Both iterative and recursive solution.**

Calculating Factorial is also a classic recursion exercise in programming. Since factorial is a recursive function, recursion becomes natural choice to solve this problem. You just need to remember formula for calculating factorial which is for  $n!$  its  $n*(n-1)*...1$ .

**10. Print following structures in Java?**

```
-----  
*  
* * *  
* * * * *  
* * *  
*  
-----  
  *  
  * * *  
* * * * *  
  * * *  
  *  
-----  
  *  
  * *  
*   *  
  * *  
  *
```

## Part – 2

---

1) Create an employee class with relevant information like name, id, salary and create employee objects.

2) Create Customer class with relevant information like name, address, account number, current balance. Create Bank Application class and add customers to the bank application with relevant methods like addCustomer, deleteCustomer, updateCustomer and getCustomerInfo etc.

3) Create Account class with account type, account number, minimum balance and current balance and provide corresponding getter and setter methods along with calInterest method. Create FixedDepositAccount, CurrAccount classes and inherit methods from Account class. Use Account class in Customer class to store account information in the customer object.

4) Create Insufficient Balance exception class and use it appropriately in Account class.

## Java Programming Assignments

5) After every 5 min query Bank Application Object and display existing customer names.