CS3101-3 Programming Language - JAVA

Fall 2004

Sept. 15th

Instructor (and TA)

- Ke Wang
- 604 CEPSR
- Web: www.cs.columbia.edu/~kewang
- Email: kewang@cs.columbia.edu, or kw2036@columbia.edu
- Tel: 212-646-6076(office)
- Office hour: Wed 2pm-4pm(temporary)

About class

- Website: http://www1.cs.columbia.edu/~kewang/cs3101/index.htm
- Meeting time and place:
  - Wed. 11am-1pm, 825 Mudd
  - Six weeks only, ends at Oct. 20th

Homework

- 5 or 6 homework
  - One homework per week
  - All programming
  - Goes out every Wed night
  - Due next Tuesday 11:59:59pm
  - Submission and HW return eletronically
  - Grade percentage to be determined
- Final?

Late policy

- You have one 24-hour extension
- Can be used only once
- Otherwise, no late homework will be accepted

Academia Integrity

- The work you submit should be implemented BY YOURSELF
- Can get help from me, or friends
- Must acknowledge all help given.

Topics to cover

- Basic Java, Objects, Classes, Inheritance, Interfaces, Exceptions, I/O
- Applets, GUI Programming, Event handling
- Multithreading, Basic Networking
- Packages, Libraries
- Some advanced topics, like collections, database, XML, etc. (If possible)

Textbook

- No required textbook
- Most of the information you need can be found online, especially at http://java.sun.com
- Tutorials and code camp: http://java.sun.com/learning/tutorial/index.html
- Java API specification: http://java.sun.com/j2se/1.4.2/docs/api/index.html
  - Important one! Should visit often when coding

Reference books

- Core Java 2, Volume I: Fundamentals
  Core Java 2, Volume II: Advanced Features
- Thinking in Java, 3rd Edition
  - Electronic version available: http://64.78.49.204/
- JAVA in a Nutshell (fourth Edition)

A word about learning a programming language

- PRACTICE
- PRACTICE
- …
- PRACTICE
- …

road map today

- Brief intro to Java
- Develop and compile environment
- A simple example
- Intro to object/class
- Java basics
- Differences from C

Intro to Java

- Java programming language
  - The one we use to write our program
  - Compiled to byte code of JVM
- Java virtual machine (JVM)
  - Java interpreter – interpret the compiled byte code
  - Software simulated CPU architecture
  - Cross-platform: support Linux, Windows, PalmOS…etc.
- Java runtime environment (JRE)
  - Predefined set of java classes available to use
  - Core Java APIs – basic utilities, I/O, graphics, network…

Java is portable,

- *As long as* there is a JVM compiled for that particular processor and OS
- Typical program, like C or C++, is compiled for a particular processor architecture and OS.
- “Write once, run everywhere!”
  - Sun’s motto for Java

Bottom line: slow but safe

- Not suitable for high-performance computation
  - Scientific computation, games, OS kernel
  - Compiled to byte codes of JVM, not native machine language instructions
  - New release of Java is improving the speed a lot
    - Just-in-time (JIT) compiler: convert byte codes to native machine language on the fly
- Very safe
  - No pointer
  - Automatic garbage collection
  - Check array access bound

Java

- Java is an *object-oriented* language, with a syntax similar to C
  - Structured around *objects* and *methods*
  - A method is an action or something you do with the object
- Avoid those overly complicated features of C++:
  - Operator overloading, pointer, templates, friend class, etc.

Getting and using java

- J2SDK freely download from http://java.sun.com
- "Your first cup of Java":
  - detailed instructions to help you run your first program
  - http://java.sun.com/docs/books/tutorial/getStarted/cupojava/index.html
- All text editors support java
  - Vi/vim, emacs, notepad, wordpad
  - Just save to .java file
- Have IDEs that comparable to Visual Studio
  - JCreator (simple)
  - Eclipse (more complicated)

Compile and run an application

- Write java class Foo containing a main() method and save in file “Foo.java”
  - The file name *MUST* be the same as class name
- Compile with: javac Foo.java
- Creates compiled .class file: Foo.class
- Run the program: java Foo
  - Notice: use the class name directly, no .class!

Hello World!

File name: Hello.java

```
/* Our first Java program – Hello.java */
public class Hello {
        //main()
        public static void main ( String[] args ) {
                System.out.println( "hello world!" );
        }
}
```

Command line arguments

Standard output, print with new line

About class

- Fundamental unit of Java program
- All java programs are classes
- Each class define a unique kind of object (a new data type)
- Each class defines a set of fields, methods or other classes
- *public*: modifier. This class is publicly available and anyone can use it.

Things to notice

- Java is *case sensitive*
- *whitespace* doesn't matter for compilation
- File name must be the same as one of the class names, *including capitalization*!
- At most one public class per file
- If there is one public class in the file, the filename must be the same as it
- Generally one class per file

What is an object?

- Object is a thing
- An object has state, behavior and identity
  - Internal variable: store state
  - Method: produce behavior
  - Unique address in memory: identity
- An object is a manifestation of a class

What is class?

- Class introduces a new data type
- A class describes a set of objects that have identical characteristics (data elements) and behaviors (methods).
  - Existing classes provided by JRE
  - User defined classes
- Once a class is established, you can make as many objects of it as you like, or none.

Simple example: class Person

- A Person has some attributes
- The class defines these properties for all people
- Each person gets his own copy of the fields
- Attributes = properties = fields

Class Person: definition

```
class Person {
    String name;
    int height; //in inches
    int weight; //in pounds
    public void printInfo(){
            System.out.println(name+" with height="+height+", weight="+weight);
    }
}
```

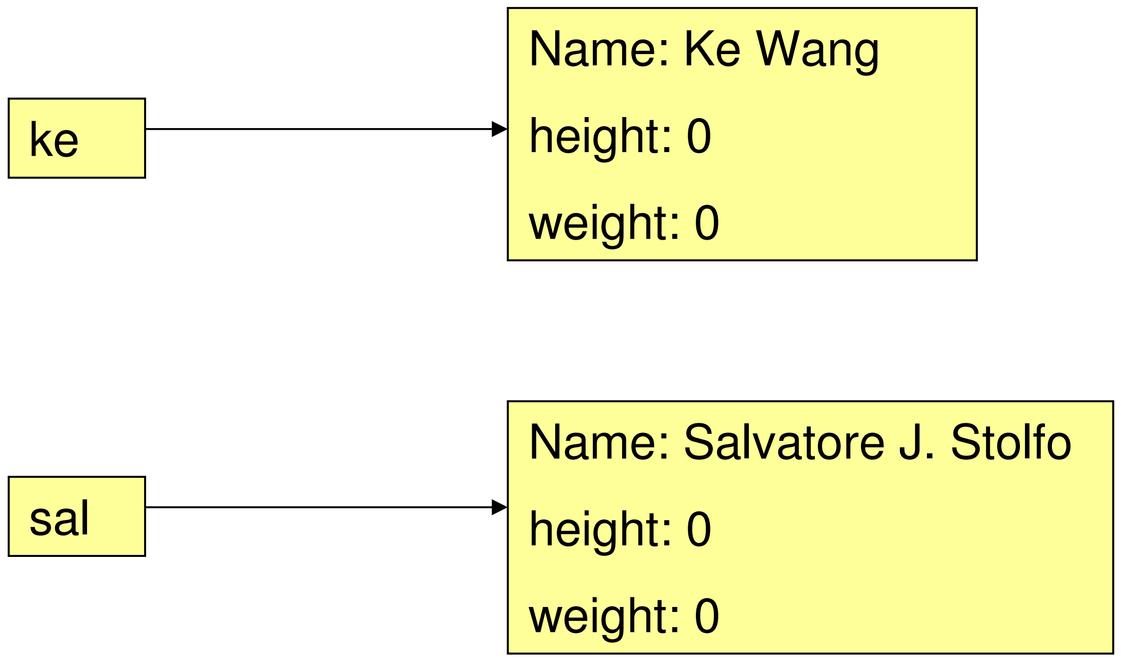
class ClassName{ /* class body goes here */ }

class: keyword

Class Person: usage

```
Person ke;      //declaration
ke = new Person();   //create an object of Person
ke.name= “Ke Wang”;   //access its field
Person sal = new Person();
sal.name=“Salvatore J. Stolfo”;
ke.printInfo();
Sal.printInfo(); // error here??
```

Class Person

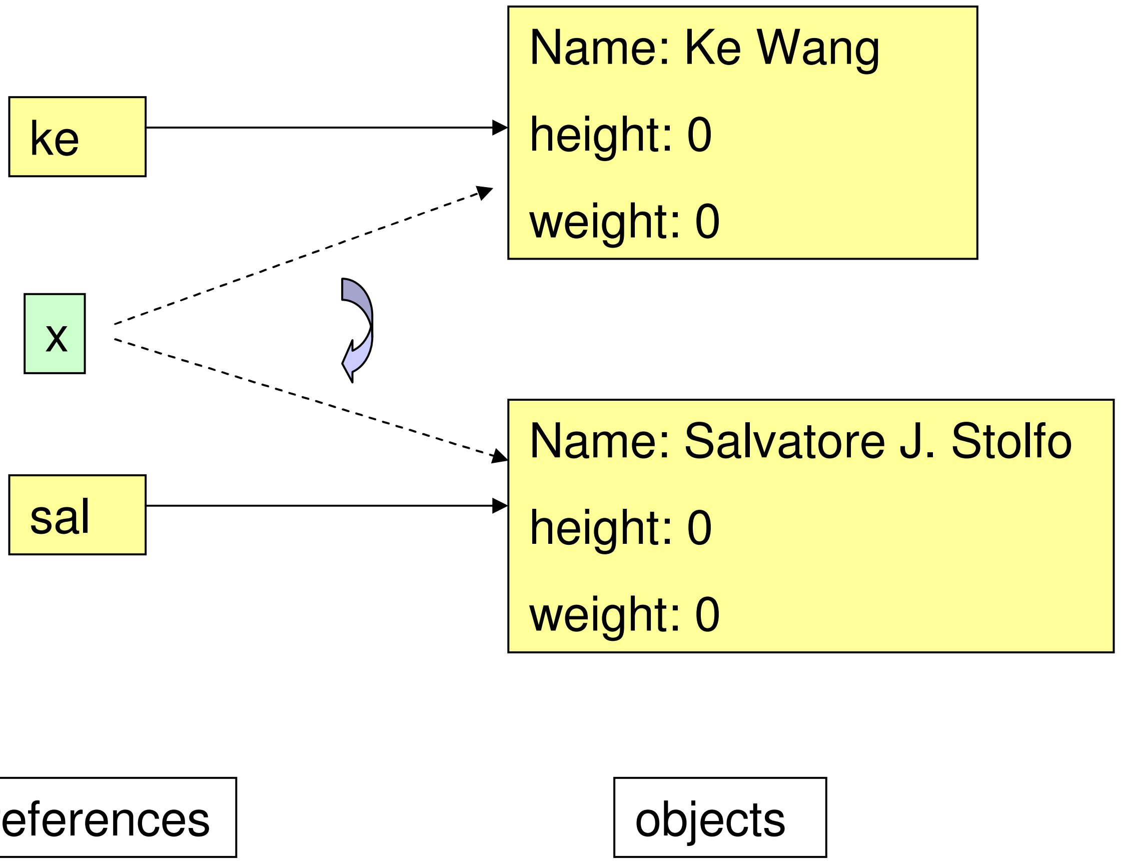


Class Person: variables

```
Person x;
x=ke;
x.printInfo();
x=sal;
x.printInfo();
```

This gives the same output as previous code !

Class Person: variables



Reference

- We call x, as well as ke and sal, “*reference*” to the object
- Handles to access an object
- Reference itself is not accessible/manipulable
  - Different from C/C++, cannot increment/decrement it
- Implemented as pointer+
  - Java runtime is watching all assignment to references
  - Why? – garbage collection (later)

Reference

```
Person ke;              //only created the reference, not an object.
                        It points to nothing now (null).

ke = new Person();      //create the object (allocate storage
                        in memory), and ke is initialized.

ke.name="Ke Wang";      //access the object through
                        the reference
```

More on reference

- Have distinguished value *null,* meaning pointing to nothing
  - if(x==null) { … }
- Multiple references can point to one object
- When no reference point to an object, that object is never accessible again.

Class Person: problem

```
ke.weight = 150; // too bad, but possible
ke.weight = -20;  // Houston, we have a problem!!
```

Need to ensure the validity of value.

Solution: ask the class to do it!

```
ke.setWeight(150);     // OK, now ke’s weight is 150
ke.setWeight(-10);     ******** Error, weight must be positive number
```

Class Person: add method

```
class Person{
    ...
    void setWeight(int w){
        if(w<=0)
             System.err.println("***** error, weight must be positive
number! ");
        else
             weight = w;
    }
}
```

Class Person: new problem

```
ke.setWeight(-10);
```

********* Error, weight must be positive number*

```
ke.weight = -20;              //haha, I'm the boss!
```

How about we forgot to use the set function? Or we just don't want to?

Solution: just make the variable inaccessible from outside!

Class Person: private variable

```
class Person{
    **private** String name;
    **private** int weight;
    **private** int height;

    **public** void setWeight(int w){
            if(w<=0)
                System.err.println("***** error, weight must be positive
number! ");
            else
                weight = w;
            }
}
```

Keyword **private**: no one can access the element except itself

Keyword **public**: everyone can access the element

Class Person

```
class Hello{

        public static void main ( String[] args ) {
                Person ke = new Person();
                ke.weight = -20;
        }
}
```

```
>javac Hello.java
Hello.java:5: weight has private access in Person
        ke.weight = -20;
          ^
1 error
```

Access functions

- Generally make fields **private** and provide **public** getField() and setField() access functions
- O-O term for this is *Encapsulation*
- C# does this by default

Java Basics: primitive types

- One group of types get special treatment in Java
- Variable is not created by “new”, not a reference
- Variable holds the value directly

Primitive types

Primitive type	Size	Minimum	Maximum	Wrapper type
boolean	1-bit	—	—	**Boolean**
char	16-bit	Unicode 0	Unicode 2^{16}- 1	**Character**
byte	8-bit	-128	+127	**Byte**
short	16-bit	-2^{15}	$+2^{15}-1$	**Short**
int	32-bit	-2^{31}	$+2^{31}-1$	**Integer**
long	64-bit	-2^{63}	$+2^{63}-1$	**Long**
float	32-bit	IEEE754	IEEE754	**Float**
double	64-bit	IEEE754	IEEE754	**Double**

Primitive types

- All numerical types are *signed*!
  - No **unsigned** keyword in Java
- The “wrapper” class allow you to make a non-primitive object to represent the primitive one
  - char c =‘a’;
  - Character C = new Character(c);
  - Character C = new Character(‘a’);

Primitive types - boolean

- boolean can never convert to or from other data type, not like C or C++
- boolean is not a integer
- if(0) doesn’t work in java
- Have to explicitly state the comparison
  - if(x ==0) {

Primitive types - char

- Char is unsigned type
- The Character wrapper class has several static methods to work with char, like isDigit(), toUpperCase() etc.

Default values for primitive members

- When a primitive type data *is a member of a class*, it’s guaranteed to get a default value even if you don’t initialize it.
- Not true for those local variables!!
  - There will be compile error if you use it without initialization

Primitive type	Default
boolean	false
char	‘\u0000’ (null)
byte	(byte)0
short	(short)0
int	0
long	0L
float	0.0f
double	0.0d

Example

```
class Hello{

        public static void main ( String[] args ) {
                int x;
                System.out.println(x);
        }
}
```

```
>javac Hello.java
Hello.java:5: variable x might not have been initialized
            System.out.println(x);
                               ^
1 error
```

Arrays in Java

- An ordered collection of something, addressed by integer index
  - Something can be primitive values, objects, or even other arrays. But all the values in an array must be of the same type.
  - Only *int* or *char* as index
  - *long* values not allowed as array index
- 0 based
  - Value indexes for array “a” with length 10
  - a[0] – a[9];
- a.length==10
  - Note: length is an attribute, not method

Arrays in Java: declaration

- Declaration
  - int[] arr;
  - Person[] persons;
  - Also support: int arr[]; Person persons[]; (confusing, should be avoided)
- Creation
  - int[] arr = new int[1024];
  - int [][] arr = { {1,2,3}, {4,5,6} };
  - Person[] persons = new Person[50];

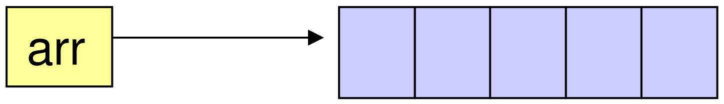
Arrays in Java: safety

- Cannot be accessed outside of its range
  - ArrayIndexOutOfBoundsException
- Guaranteed to be initialized
  - Array of primitive type will be initialized to their default value
    - Zeroes the memory for the array
  - Array of objects: actually it’s creating an array of references, and each of them is initialized to ***null.***

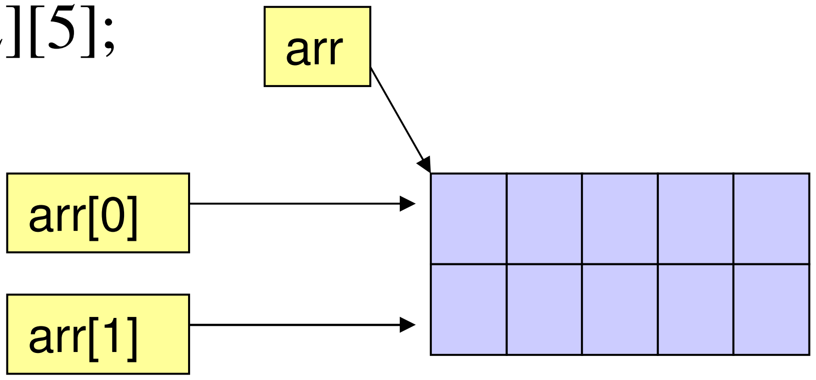
Arrays in Java:

- second kind of reference types in Java

```
int[] arr = new int [5];
```



```
int[][] arr = new int [2][5];
```



More on reference

- Java doesn’t support & address of , or *, -> dereference operators.
- reference cannot be converted to or from integer, cannot be incremented or decremented.
- When you assign an object or array to a variable, you are actually setting the variable to hold a reference to that object or array.
- Similarly, you are just passing a reference when you pass object or array to a method

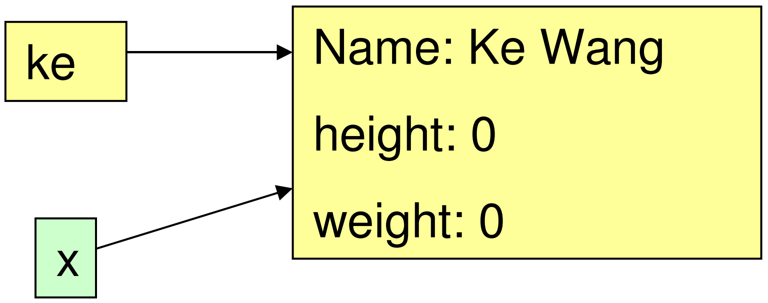
Reference vs. primitive

- Java handle objects and arrays always by reference.
  - classes and arrays are known as reference types.
  - Class and array are composite type, don’t have standard size
- Java always handle values of the primitive types directly
  - Primitive types have standard size, can be stored in a fixed amount of memory
- Because of how the primitive types and objects are handles, they behave different in two areas: copy value and compare for equality

copy

- Primitive types get copied directly by =
  - int x= 10; int y=x;
- Objects and arrays just copy the reference, still only one copy of the object existing.

```
Person ke =new Person();
ke.name="Ke Wang";
Person x=ke;
x.name="Sal";
System.out.println(ke.name);  // print Sal!
```



Compare equality

- Primitive use ==, compare their value directly

```
int x = 10; int y=10;
if(x==y) { // true !
```

- Object or array compare their reference, not content

```
Person ke =new Person();
ke.name="Ke Wang";
Person ke2 =new Person();
ke2.name="Ke Wang";
if(ke==ke2)  //false!!

Person x = ke;
if(ke==x)  //true
```

Copy objects and arrays

- Create new object, then copy all the fields individually and specifically
- Or write your own copy method in your class
- Or use the special clone() method (inherited by all objects from java.lang.Object)

```
int[] data = {1,2,3}; //an array
int[] copy = (int[]) data.clone();  //a copy of the array
```

Notice: clone() is shallow copy only! The copied object or array contains all the *primitive values and references* in the original one, but won’t clone those references, i.e., not recursive.

Compare objects and arrays

- Write you own comparison method
- Or use default *equals()* method
  - All objects inherit *equals()* from Object, but default implementation simply uses == for equality of reference
  - Check each class for their definition of *equals()*

```
String s = "cs3101";
int num=3101;
String t ="cs"+num;
if(s.equals(t)) {  //true!
```

Notice: + operator also concatenate string. If either of the operand to + is a string, the operator converts the other operand to a string

Scoping

- Scope determines both the visibility and lifetime of the names defined within the scope
- Scope is determined by the placement of {}, which is called *block*.

```
{
int x = 10;
//only x available
  {
              int y = 20;
              //both x and y available
  }
  //only x available, y out of scope!
}
```

Scoping

- Notice, you ***cannot*** do the following, although it’s legal in C/C++.

```
{
int x = 10;
  {
          int x = 20;
  }
}
```

```
Compile error
Hello.java:6: x is already defined in
main(java.lang.String[])
                    int x =20;
                        ^
1 error
```

Scope of objects

- When you create an object using **new**, the object hangs around past the end of the scope, although the reference vanishes.

```
{
            String s = new String("abc");
}
```

Reference s vanishes, but the String object still in memory

Solution: Garbage Collector!

Garbage collector

- In C++, you have to make sure that you destroy the objects when you are done with them.
  - Otherwise, memory leak.
- In Java, garbage collector do it for you.
  - It looks at all the objects created by **new** and figure out which ones are no longer being referenced. Then it release the memory for those objects.

Importing library

- If you need any routines that defined by java package

```
import java.util.*;
import java.io.*;
```

- Put at the very beginning of the java file
- java.lang.* already been imported.
- Check javadoc for the classes

Static keyword

- Want to have only one piece of storage for a data, regardless how many objects are created, or even no objects created
- Need a method that isn’t associated with any particular object of this class
- **static** keyword apply to both fields and methods
- Can be called directly by class name
  - Example: java.lang.Math
- Non-static fields/methods must be called through an instance

main()

```
class Hello{
        int num;
        public static void main(String[] args) {
                num = 10;
        }
}
```

```
>javac Hello.java
Hello.java:4: non-static variable num cannot be referenced
from a static context
          num = 10;
          ^
1 error
```

Main() doesn’t belong in a class

- Always static
  - Because program need a place to start, before any object been created.
- Poor design decision
- If you need access non-static variable of class Hello, you need to create object Hello, even if main() is in class Hello!

```
class Hello{
        int num;
        public static void main(String[] args){
                Hello h = new Hello();
                h.num = 10;
        }
}
```

Difference between C and Java

- No pointers
- No global variable across classes
- Variable declaration anywhere
- Forward reference
  - Method can be invoked before defined
- Method overloading
  - As long as the methods have different parameter lists
- No struct, union, enum type
- No variable-length argument list

Output

- System.out.println();
- System.err.println();
  - Err corresponds to Unix stderr
- System.[out|err].print();
  - Same as println(), but no terminating newline
- Easy to use, ready to go.

Input: importing library

- Need routines from java.io package
  - import java.io.*;
- System.in is not ready to use, need to build a fully functional input object on top of it
- InputStreamReader(System.in)
  - Basic byte-to-char translation
- BufferedReader(InputStreamReader isr)
  - Allows us to read in a complete line and return it as a String

```
BufferedReader in = new BufferedReader(new InputStreamReader(System.in));
//BufferedReader in = new BufferedReader(new FileReader(filename));
String line = in.readLine();
```

Basic exception

- readLine() throws IOException
- Required to enclose within try/catch block
- More on exception later

Integer.parseInt()

- Static method
- Can take the String returned by readLine() and spit out an int
- Throws NumberFormatException if String cannot be interpreted as an integer

Question?