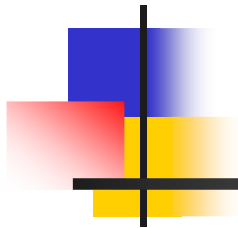
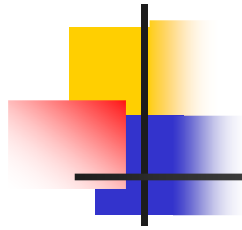


COMS W3101-1 Programming Language: Java (Spring 2013)

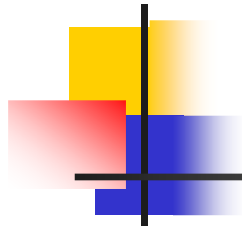


Ramana Isukapalli
ramana@cs.columbia.edu



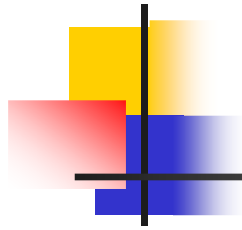
Prerequisites

- A good background in at least one programming language is **recommended**. Course overview
- Course overview - See
 - <http://www.cs.columbia.edu/~ramana>



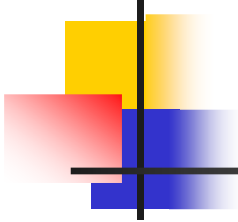
Syllabus Overview

- Java programming
 - JVM, data types, control structures, functions
- Object Oriented Programming principles of Java
 - Concepts of class/object, methods, inheritance, polymorphism, abstraction, data encapsulation
- Exception handling in Java
- Java Packages
- Generics, Collections
- Other topics.



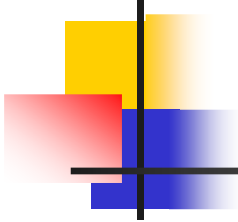
Lecture-1

- Overview of Java
 - Java programming language philosophy
 - Java virtual machine
 - Brief introduction to "class"
 - Basic data types
 - Operators
 - Control statements
 - if, for, while, do, switch/case
 - Functions



Different types of Java

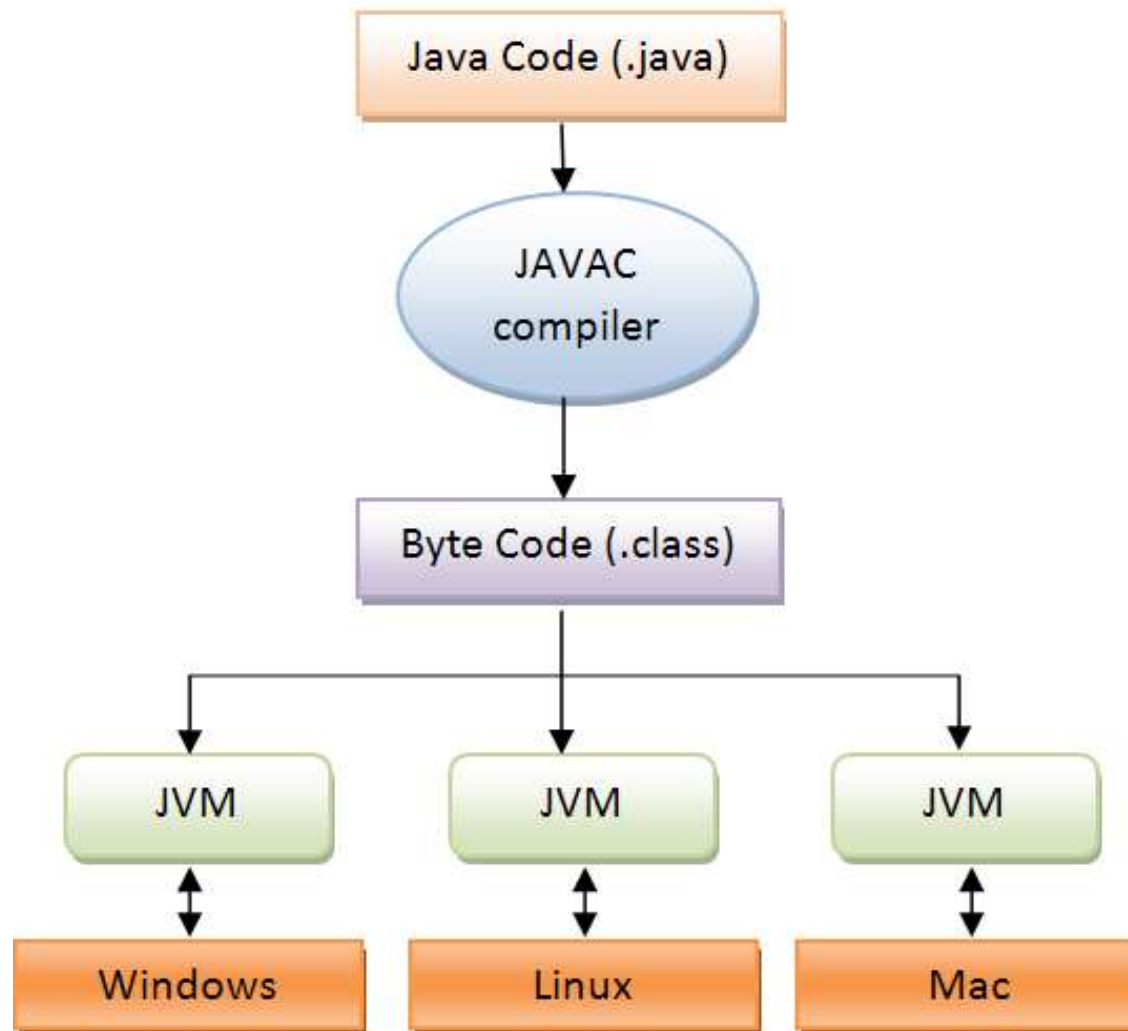
- Java "Standard Edition" (SE)
 - Basic Programming support using strings, math, I/O operations, file systems, network programming, applets, etc.
- Java "Enterprise Edition" (EE)
 - Include JDBC, RMI, servlets, JSP, etc. in addition to basic Java programming functionality.
- Java "Micro Edition" (ME)
 - Specifically designed for mobile devices applications.
- We will talk only about Java SE in this course

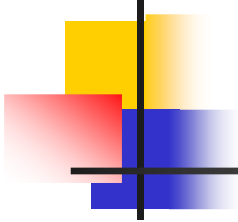


Design philosophy of Java

- Java design philosophy
 - Write Once Run Anywhere (WORA)
 - Based on "Java Virtual Machine" (JVM)
 - Uses bytecode that runs on JVM
- This is in contrast to other languages like C, C++
 - They use "Write Once Compile Anywhere" (WOCA)
 - Compiled code runs directly on the machine.

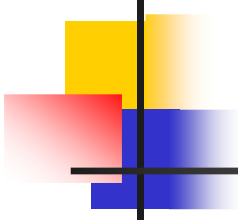
Java virtual machine





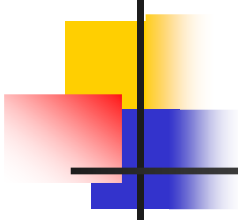
Java Virtual Machine ... contd.

- Runs on a real machine.
- Forms an intermediate layer
 - Between a real machine and the user program
- Keeps user programs independent from the machine architectures.
- A Java program generates bytecode
- Bytecode can be run on any JVM on any machine
 - Compile Once Run Anywhere



Java - classes and objects

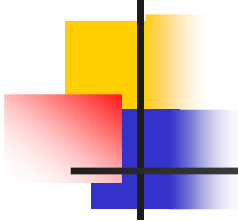
- Java is an Object Oriented Programming (OOP) language.
 - Real world entities are treated as objects.
 - objects are instances of class.
 - class are user defined.
 - Contain data and methods.
 - A class is a starting point for writing code in Java.
- We will cover details of OOP later.



A simple Java program

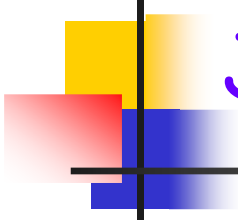
```
import java.io.*;                // Include all files under "io"
public class SimpleExample        // Define a class "SimpleExample"
{
    public static void main (String[ ] args)    // main method, program starts here
    {
        System.out.println ("Simple example"); // Just print "Simple example"
    }
}
```

- In Netbeans, just "Run" it
 - (Netbeans compiles and runs for you)
- On a shell prompt (e.g., UNIX shell)
 - Compile this using "javac SimpleJava.java"
 - Creates a file called **SimpleJava.class** - bytecode
 - Run this using "java SimpleJava"



Java programs - some rules

- The class to be run should
 - Be present in a file with the same name.
 - with a `.java` extension.
 - Be declared `public`.
 - **ONLY** the class to be run should be public.
 - This file can have other non-public classes.
 - Have a `main` method that is
 - Declared `public` and `static`.
 - Can have other functions
 - That may or may not be public, static, etc.



Java basic data types and operators

- Basic data types

- byte, char, short, int, long, float, double, boolean

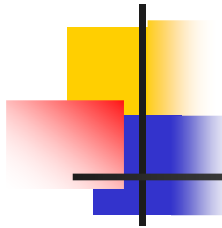
- Operators:

- Arithmetic: +, -, *, /, %, ++, --
- Logical: ==, !=, >, <, >=, <=, &&, ||, !, ?:
- Bitwise: &, |, ^, <<, >>, ~



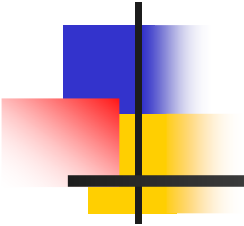
Java basic data types

Type	#bits	Default value	Min value	Max value
byte	8	0	-128	127
short	16	0	-32768	32767
int	32	0	-2147483648	2147483647
long	64	0L	-2^{63}	$(2^{63}) - 1$
float	32	0.0f		
double	64	\u0000		
Boolean	Not clearly defined	null	NA	NA
char	16	false	NA	NA

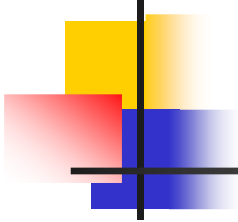


Variables

- Those used in any program to store data of any type.
 - E.g.
 - `int i = 2;        // i is a variable of type int`
 - `double j = 2.4   // j is a variable of type double`
- Value of variables can change in a program
 - `i = i + 2; // value of I is two more than
 // what it was before`



Input Output



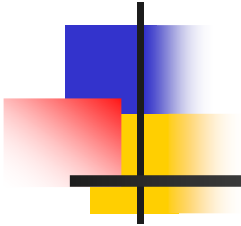
Input, Output

■ Input

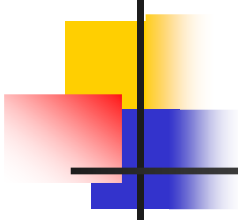
- `i = System.in.read( )` - read a character
 - `i` has the ASCII value of the character
- `BufferedReader br =`
`new BufferedReader(new InputStreamReader(System.in));`
- `String inValue = br.readLine( );`
 - Reads a string from stdin
- Others
 - We will (probably) see some more later.

■ Output

- `System.out.println` - print output to std. Output
 - E.g. `System.out.println ("any string" + variable + ...);`
- Others
 - There are others possible - we will see them later.



Control Statements



Control statements ... if

```
if (<expr_1>
{
    <body of if_expr_1>
}
else if(<expr_2>)
{
    < body of if_exp_2>
}
...
else /* default */
{
    ...
}
```

Example-1

```
if (i > j)
    System.out.println ("i is larger\n");
```

Example-2

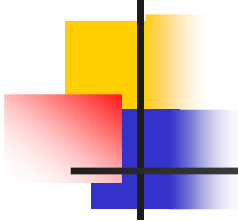
```
if (i > j)
    System.out.println ("i is larger\n");
else if ( j > i)
    System.out.println ("j is larger\n");
else
    System.out.println ("i is equal to j");
```

Example-3

```
if (i > j)
    ...
else if (i > k)
```

...

else



Control statements - for

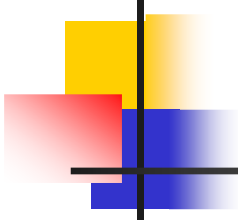
- for (<start_expr>;
 <termination_cond>;
 <loop_increment>)
{
 <body_of_for>
}

Example-1 /* print 0 to 9 */

```
for (i = 0; i < 10; i++)  
{  
    System.out.println (i);  
}
```

Example-2

```
for ( ; ; ) /* infinite loop */  
{  
    /* do something */  
}
```



Control statements - while

- Similar to for statement

- `while ( <while_cond> )`
`{`
 `<while_body>`
`}`

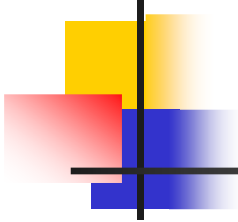
`do`
`{`
 `<body_of_do>`
`} while (condition);`

Example-1 `/* print 0 to 9 */`

```
i = 0;
while (i < 10)
{
    System.out.println (i);
    i++;
}
```

Example-2

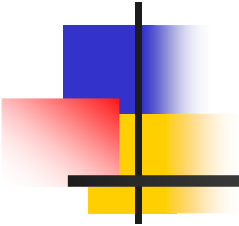
```
while (true) /* infinite loop */
{
    /* do something */
}
```



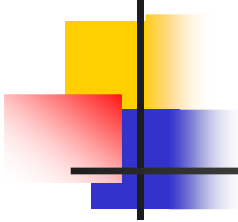
Control Statements - switch, case

```
switch (x)
{
    case val1:
        <val1_body>;
        break;
    case val2:
        <val2_body>;
        break;
    ...
    default:
        <default_body>
}
```

```
int x = 2;
switch (x)
{
    case 1:
        function1( );
        break;
    case 2:
        function2( ); /* executed */
        break;
    ...
    default:
        default_function( );
}
```



Functions

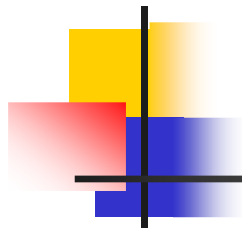


Java functions

- A group of statements
 - To perform a task
 - Possibly return a value
- Unlike C, functions are part of a class in Java

- Syntax

```
<return_type> fn_name (arguments)
{
    // function body
}
```



Java functions ... example

```
import java.io.*;

public class anyClass                                // Define a class first
{
    int square (int x)                                // function to compute square
    {
        return (x * x);
    }

    public static void main(String args[ ]) //Starting point of the program
    {
        int i = 5;
        anyClass ac = new anyClass();                // Create an object first
        int i_sq = ac.square (5);                     // Call its function
        System.out.println ("Square of 5 is: " + i_sq);
    }
}
```