Java Features

- object-oriented, imperative
- C/C++-like (C++ ++ --)
- safe, robust, multithreaded, architecture independent: a major goalwas to be able to distribute compiled code over the world-wide-web, but this has been less successful than Java's creators hoped.
- no pointers (only implicit references)
- no undefined or architecture dependent features: this is veryimportant if code is to run on many different kinds of computers andthe language is to be implemented by many different groups. For example,a Java `int` is always 32-bit two's complement. In C, the meaning of `int` depends on the implementation: it might be 8, 16, 32, or 64 bits.
- static scoping, static typing, C-like precedence and associativity,short circuit evaluation: you will find Java's low-levelsyntax quite familiar after C.
- no *operator* overloading (but does have *method* overloading),
- eager (left to right) evaluation of parameters,
- strongly typed, hybrid implementation with a virtual machine,
- run-time memory mangament: explicit allocation and implicit deallocation (garbage collection)
- heap-dynamic objects (only), reference types (all objects),no dangling pointers, no memory leaks.

The Java Virtual Machine (JVM)

A Java source program (e.g., Foo.java) is compiled to a *class* file (Foo.class) containing machine instructionscalled*byte codes* for the *Java Virtual Machine*. Compilationis performed by the `javac` program -- the Java compiler.Class files are (supposed to be) portable across different implementations of the JVM. To run a class file use the `java` program to startthe JVM executing your class file.

There aredifferent implementations of JVM for different computer architecturesand operating systems. Different implementors of JVMs have different goalsfor performance as well.

The advantages of compiling code for the JVM instead of native codefor a particular computer include:

- The JVM has much more safety checking at the byte-code level thanmost hardware architectures have for their machine instructions. This is importantif you are going to accept code sent to you over the net.
- Byte-codes may be more compact than machine code for some kinds of hardware (especiallyso-called RISC hardware).
- The class file design supports dynamic loading and linking of code which wasa much bigger deal when Java was created than it is now when many of these ideasare common in so-called dynamic libraries for C and C++.

The primary disadvantage of byte-codes is that the JVM must execute severalmachine instructions to interpret each byte-code instruction. However, techniquessuch as just-in-time compilation (JIT) have been developed that allow the JVM to translate byte-code to native code and save it for re-use. Much effort has gone intoJIT technology over the last 10 years.

Java's approach, using JVM, can also be contrasted to the Python approach whichalso uses byte codes. Unlike Java, Python does not separate the compiler and virtualmachine into separate programs: there is just the `python` command. Python translates python source code files to byte code and then executes what it hasjust compiled. It also may save a copy of the compiled code for `Foo.py`in file `Foo.pyc` to save itself the trouble of recompiling. But this is just an optimization. There is no attempt in Python to make the compiled filesportable between implementations or to ensure that it is safe to accept a compiledfile that you got from someone else.

C/C++ features not in Java and what to do instead

- structures: use objects. Objects are not required to have methods--they can alsojust package together data.
- unions: there are two cases. If the union is really a union of high-level types, forexample to allow a container to hold objects of different types, you can use objectswith a common superclass. If the C union is being used to let you look at bit patternsof data of one type as if it were another type there is no way to do it easily in Java.In general such code is unsafe.
- functions: use static methods of a class
- function pointer parameters: use object parameters
- multiple inheritance: for multiple subtyping use Java *interfaces* which resembleclasses without any associated implementation. Because there is no implementation thedifficulties that arise with multiple inheritance are avoided.

- operator overloading: can't do it
- pointers: Java objects and arrays are always represented by *references*which are pointer-like in some ways but do not support pointer arithmetic (which would be unsafe). There is no way to create pointers to values of base types like int, char, boolean, float.

Hello World

```
import java.io.*// Helloworld.java//   compile using 'javac Helloworld.java'// This is a hello world e
```

Note that every program must have at least one class. By defaultthe `main` method is called when Java first runs.`main` must be static and public."static" means it stays with the class (not an object method, buta class method). static methods can be invoked using a classqualification, e.g., the following calls the main method in theHelloworld class.

```
Helloworld.main()
```

By convention, the class `Helloworld` must be in thefile `Helloworld.java`.

Compiling and Running Java

Java code is complied into JVM (Java Virtual Machine) code by theJava compilier (`javac`).

```
javac Helloworld.java
```

will produce `Helloworld.class`.If `Helloworld.java` imports another class, that classwill be automatically compiled as well.Standard libraries are automatically imported (and are precompiled).In the above program, `java.io.*` is a standard library.To run the resulting program, start the JVM on the class containingthe `main` method, e.g.,

```
java Helloworld
```

will start with `main` in the Helloworld class.

Comments

Two varieties

```
// I am one line comment /* I am a        multi-line comment */
```

Classes

- a class is a prototype
- typical example

```
public class Name {  public int method1(arg_type arg, ...) { }  public static int method2(arg_type
```

- static methods remain with class

Javadoc

Javadoc is the Java API Documentation system.It can produce HTML output of the documentation in your program.

```
/*** TauZamanSystem is the entrance point for * TauZaman services. It contains methods, which provide*
```

Data Types

- similar to C

```
 short   16-bit two's complement   Short integer int     32-bit two's complement   Integer long
```

- reference types - Strings, arrays, classes, interfaces
- no pointers, struct, or unions

Variable Declarations

- similar to C: **type name**

```
int i;  String s;     // a string is an object  float x, j;
```

- initialization similiar to C

```
int i = 3;
```

- scope is within declaring block

Java Operators

C-like

```
This table summarizes Java's binary arithmetic operations:   Operator    Use        Description    +
```

Flow of Control

C-like

```
  if (x == 34) {... } else {...}  for (i = 0; i < length; i++) {...}  switch (month) {    case 1:  Syst
```

Arrays

Arrays are objects

```
   int[] arrayOfInts = new int[10]   for (int j = 0; j < arrayOfInts.length; j ++) {     arrayOfInts[j]
```

Strings

A string is an object.

- String is immutable/StringBuffer is mutable

```
String joe = new String("hello how are you");  // Alternatively  String joe = "hello how are you
```

- + is concatenation, automatic conversion

```
count = 23;  System.out.println("Input has " + count + " chars.");
```

I/O - output

```
import java.io.*;class PersonalHello {  public static void main (String args[])    {      byte name[] =
```

I/O - input

```
  static int getNextInteger() {      String line;      DataInputStream in = new DataInputStream(System.
```