The Java Environment

Learning objectives

- Understand the basic features of Java
  - What are portability and robustness?
- Understand the concepts of bytecode and interpreter
  - What is the JVM?
- Learn few coding conventions
  - How shall I name identifiers?

Java Timeline

- 1991: SUN develops a programming language for cable TV set-top boxes
- Simple, OO, platform independent
- 1994: Java-based web browser (HotJava), the idea of “applet” comes out
- 1996: first version of Java (1.0)

See also: http://oracle.com.edgesuite.net/timeline/java/

Java timeline (cont' d)

- 1996: Netscape supports Java
  - Popularity grows
  - Java 1.02 released, followed by many updated releases in close rounds
- 1997: Java 1.1 released, major leap over for the language
- 1998: Java 2 platform (v. 1.2) released (libraries)
- 2000: J2SE 1.3 (platform enhancements, HotSpot)
- 2002: J2SE 1.4 (several new APIs), e.g.
  - XML
  - Logging

Java timeline (cont' d)

- 2005: J2SE 5.0 (Language enhancements)
  - Generics
- 2006: Java SE 6 (Faster Graphics),
  - goes open source
- 2011: Java SE 7 (I/O improvements)
- 2014: Java SE 8 (Language evolution)
  - Lambda expressions
  - Functional paradigm
  - New features marked with ☕

OO language features

- OO language provides constructs to:
  - Define classes (types) in a hierarchic way (inheritance)
  - Create/destroy objects dynamically
  - Send messages (w/ dynamic binding)
- No procedural constructs (pure OO language)
  - no functions, class methods only
  - no global vars, class attributes only

Java features

- Platform independence (portability)
  - Write once, run everywhere
  - Translated to intermediate language (bytecode)
  - Interpreted (with optimizations, e.g. JIT)
- High dynamicity
  - Run time loading and linking
  - Dynamic array sizes
- Automatic garbage collection

Java features (cont’d)

- Robust language, i.e. less error prone
  - Strong type model and no explicit pointers
    - Compile-time checks
  - Run-time checks
    - No array overflow
  - Garbage collection
    - No memory leaks
  - Exceptions as a pervasive mechanism to check errors

Java features (cont’d)

- Shares many syntax elements w/ C++
  - Learning curve is less steep for C/C++ programmers
- Quasi-pure OO language
  - Only classes and objects (no functions, pointers, and so on)
- Basic types deviates from pure OO...
- Easy to use

Java features – Classes

- There is only one first level concept: the class

```
public class First {
}
```

- The source code of a class sits in a *.java* file having the *same name*
    - Rule: one file per class
    - Enforced automatically by IDEs
    - Case-wise name correspondence

Java features – Methods

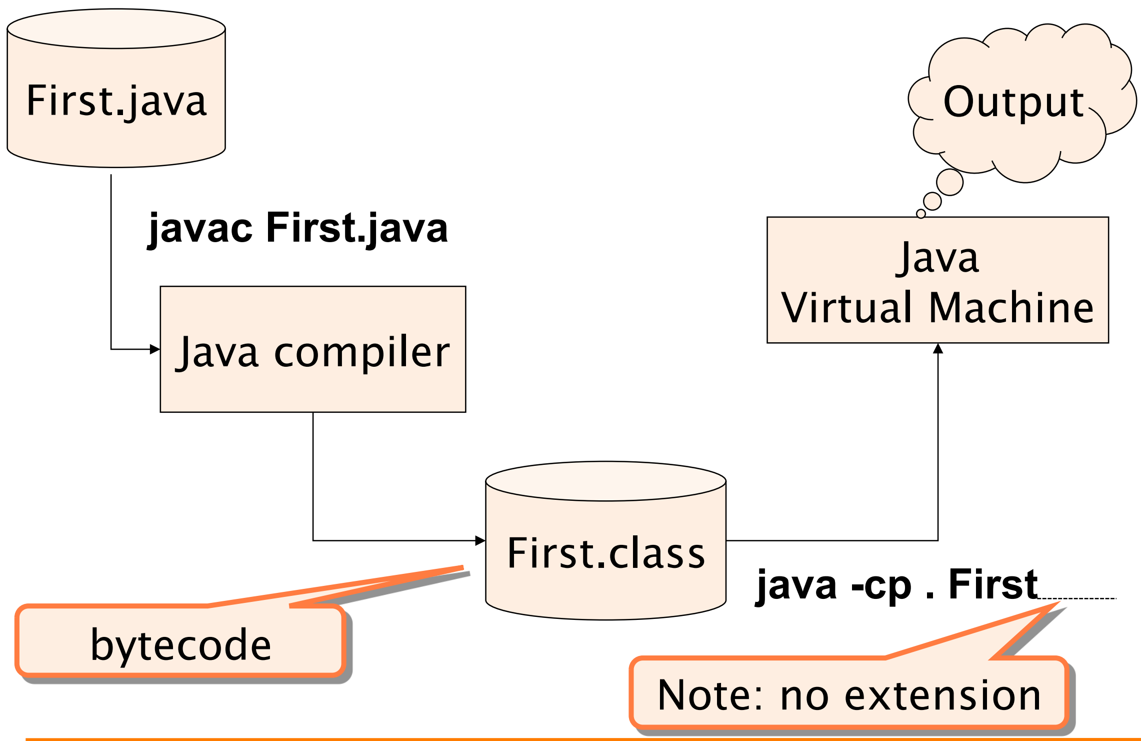
- In Java there are no functions, but only methods within classes
- The execution of a Java program starts from a special method:

```
public static void main(String[] args)
```

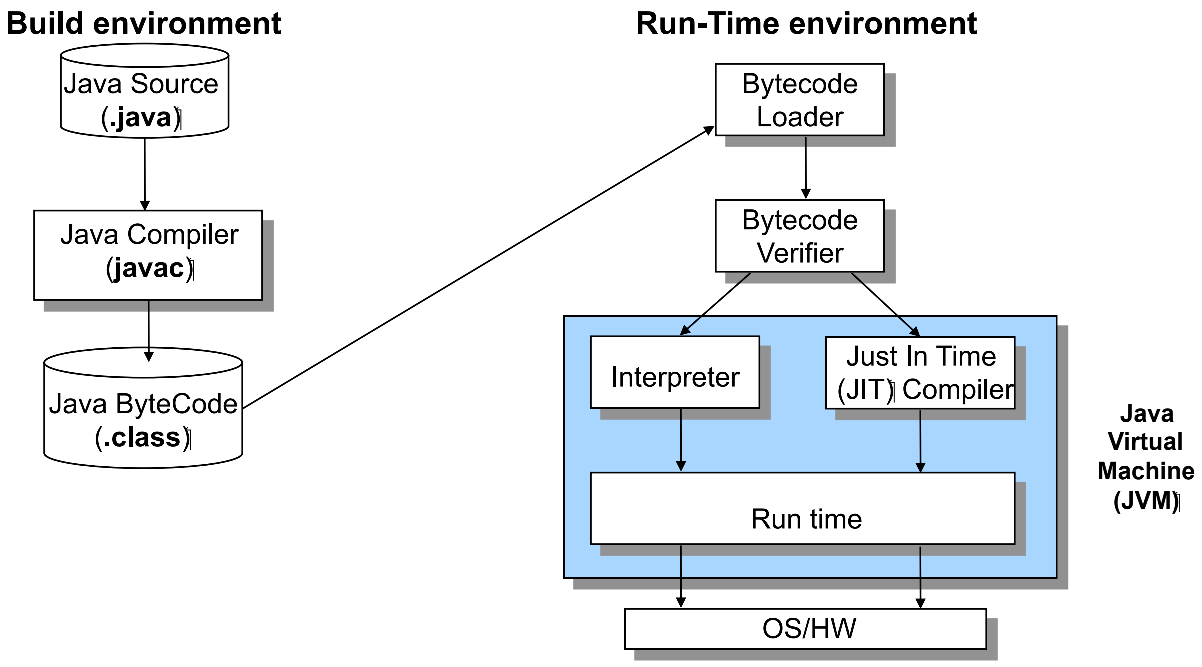
In C we find the function:
```
int main(int argc, char* argv[])
```

- Note
    - return type is **`void`**
    - **`args[0]`** is the first argument on the command line (after the program name)

Build and run



Building and running



Java Ecosystem

- Java language
- Java platform
  - JVM
  - Class libraries (API)
  - SDK

Dynamic class loading

- JVM loading is based on the classpath:
  - list of locations whence classes can be loaded
- When class X is required:
  - For each location in the classpath:
    - Look for file X.class
    - If present load the class
    - Otherwise move to next location

Example

- File: First.java:

```
public class First {
  public static void main(String[] args){
    int a;
    a = 3;
    System.out.println(a);
  }
}
```

Java features (cont'd)

- Supports "programming in the large"
  - JavaDoc
  - Class libraries (Packages)
- Lots of standard utilities included
  - Concurrency (thread)
  - Graphics (GUI) (library)
  - Network programming (library)
    - socket, RMI
    - applet (client side programming)

Types of Java programs

- Application
  - It's a common program, similarly to C executable programs
  - Runs through the Java interpreter (**`java`**) of the installed Java Virtual Machine

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

Types of Java programs

- Applet (client browser)
  - Java code dynamically downloaded
  - Execution is limited by "sandbox"
- Servlet (web server)
  - In J2EE (Java 2 Enterprise Edition)
- Midlet (mobile devices)
  - In J2ME (Java 2 Micro Edition)
- Android App (Android device)
  - Java

Java development environment

- Java SE 8 (http://www.oracle.com/technetwork/java/javase)
  - **javac** compiler
  - **jdb** debugger
  - **JRE** (Java Run Time Environment)
    - JVM
    - Native packages (awt, swing, system, etc)
- Docs
  - http://docs.oracle.com/javase/
- Eclipse:
  - Integrated development environment (IDE)
  - http://www.eclipse.org/

Coding conventions

- Use camelBackCapitalization for compound names, not underscore
- Class name must be capitalized
- Method names, object instance names, attributes, method variables must all start in lowercase
- Constants must be all uppercases (w/ underscore)
- Indent properly

Coding conventions (example)

```
class ClassName {

     final static double PI = 3.14;

     private int attributeName;

     public void methodName {
               int var;
               if ( var==0 ) {
               }
          }
}
```

Deployment – Jar

- Java programs are packaged and deployed in jar files.
- Jar files are compressed archives
  - Like zip files
  - Contain additional meta-information
- It is possible to directly execute the contents of a jar file from a JVM
  - JVM can load classes from within a JAR

Jar command

- A jar file can be created using:

```
jar cvf my.jar *.class
```

- The contents can be seen with:

```
jar tf my.jar
```

- To run a class included in a jar:

```
java -cp my.jar First
```

  - The “–cp my.jar” option adds the jar to the JVM classpath

Jar Main class

- When a main class for a jar is defined, it can executed simply by:

```
java -jar my.jar
```

- To define a main class, a manifest file must be added to the jar with:

```
jar cvfm my.jar manifest.txt
```

```
Main-Class: First
```

FAQ

- Which is more “powefull”: Java or C?
  - Performance: C is better though non that much better (JIT)
  - Ease of use: Java
  - Error containment: Java
- How can I generate an “.exe” file?
  - You cannot. Use an installed JVM to execute the program
  - GCJ: http://gcc.gnu.org/java/

FAQ

- I downloaded Java on my PC but I cannot compile Java programs:
  - Check you downloaded Java SDK (including the compiler) not Java RTE or JRE (just the JVM)
  - Check the path includes *pathToJava*/bin
- Note: Eclipse uses a different compiler than javac

FAQ

- Java cannot find a class (ClassNotFoundException)
  - The name of the class must not include the extension .class:
    - Es. java First
  - Check you are in the right place in your file system
    - java looks for classes starting from the current working directory

Wrap-up session

- Java is a quasi-pure OO language
- Java is interpreted
- Java is robust (no explicit pointers, static/ dynamic checks, garbage collection)
- Java provides many utilities (data types, threads, networking, graphics)
- Java can used for different types of programs
- Coding conventions are not “just aesthetic”