The Java programming environment

Cleaned up version of C++:
- no header files, macros, pointers and references, unions, structures, operator overloading, virtual base classes, templates, etc.

Object-orientation: Classes + Inheritance

Distributed: RMI, Servlet, Distributed object programming.

Robust: Strong typing + no pointer + garbage collection

Secure: Type-safety + access control

Architecture neutral: architecture neutral representation

Portable

Interpreted
- *High performance through* Just in time compilation + runtime modification of code

Multi-threaded

Dynamic

The Java programming environment

Java programming language specification
- Syntax of Java programs
- Defines different constructs and their semantics

Java byte code: Intermediate representation for Java programs

Java compiler: Transform Java programs into Java byte code

Java interpreter: Read programs written in Java byte code and execute them

Java virtual machine: Runtime system that provides various services to running programs

Java programming environment: Set of libraries that provide services such as GUI, data structures,etc.

Java enabled browsers: Browsers that include a JVM + ability to load programs from remote hosts

Java features

Names and scopes

Simple data types: integers, floats, char, strings

Control statements: if-then-else, switch, while, for

Classes and objects

Inheritance

Interfaces

Exceptions

Concurrency

Packages

Name spaces

Reflection

Applet model

Java: A tiny intro

How are Java programs written?

How are variables declared?

How are expressions specified?

How are control structures defined?

How to define simple methods?

What are classes and objects?

What about exceptions?

What about concurrency?

How are Java programs written?

Define a class HelloWorld and store it into a file:
HelloWorld.java:

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

Compile HelloWorld.java

```
Javac HelloWorld.java
```

Output: HelloWorld.class

Run

```
Java HelloWorld
```

Output: Hello, World

How are variables declared?

Fibonacci:

```
class Fibonacci {
  public static void main(String[] arg) {
    int lo = 1;
    int hi = 1;
    System.out.println(lo);
    while (hi < 50) {
          System.out.println(hi);
          hi = lo + hi;
          lo = hi – lo;
    }
  }
}
```

How to define expressions?

Arithmetic: +, -, *,/, %, =

```
8 + 3 * 2 /4
```

Use standard precedence and associativity rules

Predicates: ==, !=, >, <, >=, <=

```
public class Demo {
  public static void main (String[] argv) {
      boolean b;
      b = (2 + 2 == 4);
      System.out.println(b);
  }
}
```

How are simple methods defined?

Every method is defined inside a Java class definition

```
public class Movie {
  public static int movieRating(int s, int a, int d) {
      return s+a+d;
  }
}
public class Demo {
  public static void main (String argv[]) {
      int script = 6; acting = 9; directing = 8;
      displayRating(script, acting, directing);
  }
  public static void displayRating(int s, int a, int d) {
      System.out.print("The rating of this movie is");
      System.out.println(Movie.movieRating(s, a, d);
  }
}
```

How are control structures specified?

Typical flow of control statements: if-then-else, while, switch, do-while, and blocks

```
class ImprovedFibo {
    static final int MAX_INDEX = 10;
    public static void main (String[] args) {
      int lo = 1;
      int hi = 1;
      for (int I = 2; I < MAX_INDEX; I++) {
                if (I %2 = 0)
                          mark = " *";
                else mark = "";
                System.out.println(I+ ": " + hi + mark);
                hi = lo + hi;
                lo = hi – lo;
      }
    }
}
```

What are classes and objects?

Classes: templates for constructing instances

- Fields
  - Instance variables
  - Static variables
- Methods
  - Instance
  - Static

```
class Point {
    public double x, y;
}
Point lowerleft = new Point();
Point upperRight = new Point();
Point middlePoint = new Point();
lowerLeft.x = 0.0; lowerLeft = 0.0;
upperRight.x = 1280.0; upperRight.y = 1024.0
middlePoint.x  = 640.0; middlePoint.y = 512.0
```

How are instance methods defined?

Instance methods take an implicit parameter: instance on which method is invoked

```
public class Movie {
      public int script, acting, directing;
      public int rating() {
            return script + acting + directing;
      }
}
public class Demo {
      public static void main (String argv[]) {
            Movie m = new Movie();
            m.script = 6; m.acting = 9; m.directing = 8;
            System.out.print("The rating of this movie is");
            System.out.println(m.rating());
      }
}
```

How to extend classes?

Inheritance: mechanism for extending behavior of classes; leads to construction of hierarchy of classes (More detail in class)

What happens when class C extends class D:

- Inherits instance variables
- Inherits static variables
- Inherits instance methods
- Inherits static methods
- C can:
  - Add new instance variables
  - Add new methods (static and dynamic)
  - Modify methods (only implementation)
  - Cannot delete anything

How to extend classes?

```
public class Attraction {
        public int minutes;
        public Attraction() {minutes = 75;}
        public getMinutes() {return minutes;}
        public setMinutes(int d) {minutes = d;}
}
public class Movie extends Attraction {
        public int script, acting, directing;
        public int Movie() {script = 5; acting = 5; directing = 5;}
        public int Movie(int s, a, d) {
                script = s; acting = a; directing = d;
        }
        public int Rating() {return script + acting + directing;}
}
public class Symphony extends Attraction {
        public int playing, music, conducting;
        public int symphony() {playing = music = conducting = 5;}
        public int symphony(int p, m, c) {
        playing = p; music = m; conducting = c;
        }
        public int rating() {return playing + music + conducting;}
}
```

What are abstract classes?

Abstract class: Merely a place holder for class definitions; cannot be used to create instances.;

```
public abstract class Attraction {
        public int minutes;
        public Attraction() {minutes = 75;}
        public getMinutes() {return minutes;}
        public setMinutes(int d) {minutes = d;}
}
```

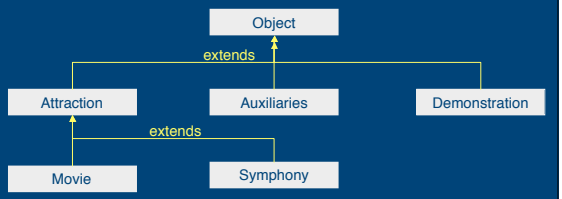
Following is an error:

```
Attraction x;
x = new Attraction();
```

Following is not an error:

```
public class Movie extends Attraction { ... }
public class Symphony extends Attraction { ... }
Attraction x;
x = new Movie ();
x = new Symphony();
```

Packages



- How do organize above classes into a single unit?
  - Put them in file? However, only one public class/file (whose name is same as that of file)
- Solution: Place several files (compilation units) into a package

Packages – cont'd.

units of organizing related Classes, Interfaces, Sub packages

Why?

- Reduce name clashing
- Limit visibility of names

Java programs typically organized in terms of packages and subpackages

- Each package may then be divided into several packages, subpackages, and classes
- Each class can then be stored in a separate file

Each source file starts with something like:

```
package mypackage
```

- Code in source file is now part of mypackage

Packages – cont'd.

```
package onto.java.entertainment;
public abstract class Attraction { … }
```

```
package onto.java.entertainment;
public class Movie extends class Attraction {…}
```

```
package onto.java.entertainment;
import java.io.*;
import java.util.*;
public class Auxiliaries { … }
```

- Where to store packages?
- How does Java find packages?
- Export and Import
- Access control