Java

A nicer version of C++, but not as purely OO as Smalltalk
- most data structures and values are objects
- most operations are methods invoked by dynamically dispatched messages
- uniform reference data model, with garbage collection
  - no explicit pointers, as they’re all implicit
- strongly, statically typed
- subtype polymorphism, but no parametric polymorphism
- no first-class functions, but has anonymous, inner classes
- C-style control structures and syntax

Includes rich standard data structure & graphics libraries

Includes interactive graphical programming environment

Based on portable virtual machine (like Smalltalk)

An example

```java
package Geometry.Points;

public interface POINT {
    public int x();
    public int y();
    public POINT add(POINT p);
};

public abstract class Point implements POINT {
    public abstract int x();
    public abstract int y();
    public POINT add(POINT p) {
        return new CartPoint(x() + p.x(), y() + p.y());
    }
    public String toString() {
        return "(" + x() + "," + y() + ");"
    }
}

public class CartPoint extends Point {
    protected int _x, _y;
    public CartPoint(int x, int y) {
        _x = x; _y = y;
    }
    public CartPoint() { this(0, 0); }
    public int x() { return _x; }
    public int y() { return _y; }
}

More of the example

```java
public interface POINT3D extends POINT {
    public int z();
    public POINT3D add(POINT3D p);
};

public abstract class Point3D extends Point implements POINT3D {
    public abstract int z();
    public POINT3D add(POINT3D p) {
        return new CartPoint3D(
            x() + p.x(), y() + p.y(), z() + p.z());
    }
    public String toString() {
        return "("+x()+","+y()+","+z()+")";
    }
}

public class CartPoint3D extends CartPoint3D implements POINT3D {
    protected int _z;
    public CartPoint3D(int x, int y, int z) {
        super(x, y); _z = z;
    }
    public CartPoint3D() { this(0, 0, 0); }
    public int z() { return _z; }
    public POINT3D add(POINT3D p) { as above }
    public String toString() { as above }
}
```

A client

```java
import Geometry.Points.*;

public class Client {
    public static void main(String[] args) {
        POINT p1 = new CartPoint(3, 4);
        POINT p2 = new CartPoint(6, 7);
        POINT p3 = p1.add(p2);
        System.out.println(p3.toString());

        POINT3D q1 = new CartPoint3D(3, 4, 5);
        POINT3D q2 = new CartPoint3D(6, 7, 8);
        POINT3D q3 = q1.add(q2);
        System.out.println(q3.toString());

        POINT r1 = q1;
        POINT r2 = q2;
        POINT r3 = r1.add(r2);
        System.out.println(r3.toString());
    }
}
```
Static overloading vs. dynamic dispatching

Some functions in or inherited by CartPoint3D:
CartPoint3D(int x, int y, int z)
CartPoint3D()
POINT add(POINT p) // inherited from Point
POINT3D add(POINT3D p) // in CartPoint3D
String toString() // inherited from Point
String toString() // in CartPoint3D

Distinguish based on # of arguments, and static types of arguments
POINT3D p3d = ...;
POINT p = p3d;
p3d.add(p3d); // invokes POINT3D add(POINT3D p)
p3d.add(p); // invokes POINT add(POINT p)

By contrast, overridden methods selected based on dynamic class of receiver
p3d.toString(); // invokes toString in CartPoint3D
p.toString(); // invokes toString in CartPoint3D

Can do both simultaneously