Polymorphism through Inheritance and Virtual Methods

- Sidebar: How C++ allocates objects
  - Point p = Point(1,2,3,4);
  - The point object is allocated on the stack
  - Point* p = new Point(1,2,3,4);
  - The point object is allocated on the heap; the stack only contains the pointer
  - Java: Point p = new Point(1,2,3,4);
  - The point object is allocated on the heap; there is no way to allocate objects on the stack

- Polymorphism ("many shapes") and virtual methods
  - Consider: Object o = new Point(1,2,3,4); System.out.println(o.toString());
    - The assignment is legal b/c Point is a subclass of Object
    - Instances of point can be used anywhere Object is declared
    - o's static type is java.lang.Object
      - It determines what you may do with o
      - The compiler enforces that only the public fields and methods of Object are accessed
        - o.distance(Point.ORIGIN) is ILLEGAL b/c distance() is defined by Point
    - o's dynamic type is xtc.oop.Point
      - It determines what actually happens with o
      - So, when invoking o.toString(), the method defined by Point is executed
        - This kind of method invocation is called "virtual dispatch"
        - Point.toString() "overrides" the virtual method "Object.toString()"
      - In general: Start with the dynamic type and look for a matching method. If found, execute it. If not found, look for a matching method in the superclass. If found, execute it. And so on.
        - The method is guaranteed to exist b/c the compiler enforces compliance with the static type, i.e., you can only call methods defined by the static type
  - Java: all public and protected methods are virtual
  - C++: the superclass must explicitly mark a method as virtual
  - Virtual dispatch only works with pointers and references to objects