Advanced Programming Techniques

Advanced Java Survey

Christopher Moretti

So you think you know C ...

```
int main(void) {
  int n=2;
  printf("%d %d %d", n++, n, ++n);
}
```

```
int main(void) {
  int x=4;
  printf("%d", ++x*++x);
}
```

```
int main(void) {
   int a=5;
   printf("%d %d %d", a++, a++, a++);
   a=5;
   printf("\n%d %d %d", a, a++, ++a);
}
```

Well, at least there's Java ...

```
private static void n() {
  int n = 2;
  StdOut.printf("%d %d %d\n", n++, n, ++n);
  n = 2;
  StdOut.printf("%d %d %d\n", ++n, n, n++);
}
private static void x() {
  int x = 4;
  StdOut.printf("%d\n", ++x*++x);
}
private static void a() {
  int a = 5;
  StdOut.printf("%d %d %d\n", a++, a++, a++);
  a=5;
  StdOut.printf("%d %d %d\n", a, a++, ++a);
}
```

```
==========n==========
2 3 4
3 3 3

==========x==========
30

==========a==========
5 6 7
5 5 7
```

Java History

- Invented primarily by James Gosling (Sun Microsystems)
- 1990 - Oak language for embedded systems
  - Implemented as interpreter with virtual machine
- 1993 - Java rebranding for the web
  - Java Virtual Machine (JVM) runs in a browser
- 1994 - Netscape Navigator supports Java
- 1997 - early 2000's - Sun continually sues Microsoft

Microsoft move actually good for Java?

Software giant could dramatically boost the number of Java programmers.

By Chris Nerney

The reaction of Java true believers to Microsoft Corp.'s announcement last week of a Windows-only version of the programming language ranged from anger to, well, comparisons to the fight against dictatorship in World War II.

But some vendors and analysts said the beta release of Microsoft's Visual J++ 6.0, the new version of its popular Java development tool, will dramatically boost the number of Java programmers, and is, therefore, a step forward.

"Microsoft's actions are great news for Java," said David Litwack, founder of SilverStream Software, Inc., a start-up that makes a Java-based Web application platform.

Many developers, however, said Microsoft's intent is to undermine Java's most attractive feature: Its promise of write-once, run-anywhere platform independence.

Visual J++ 6.0 includes code that restricts applications written with it to the Windows platform. And new Windows Foundation Classes work only with Windows functions and services.

IBM, a staunch partner of Sun Microsystems, Inc., the developer and licensee of Java technology, released a statement blasting Microsoft for its alleged "crusade against Java."

"[Visual J++ 6.0] is yet another attempt by Microsoft to tie Java developers to Windows through platform-specific extensions," the statement said.

The Java Lobby, a group formed last August to give Java developers a voice in the growing battle between Microsoft and Sun for control of the language, went even further. "Like World War II, this is a battle in which the forces of freedom must align against the aggressive forces of tyranny," wrote Rick Ross, Java Lobby founder, in an editorial posted on the group's Web site. "Microsoft is bent upon controlling the future of technology and on limiting our options only to those which work to their economic advantage. From this point forward we must recognize that we are, in fact, at war with Microsoft."

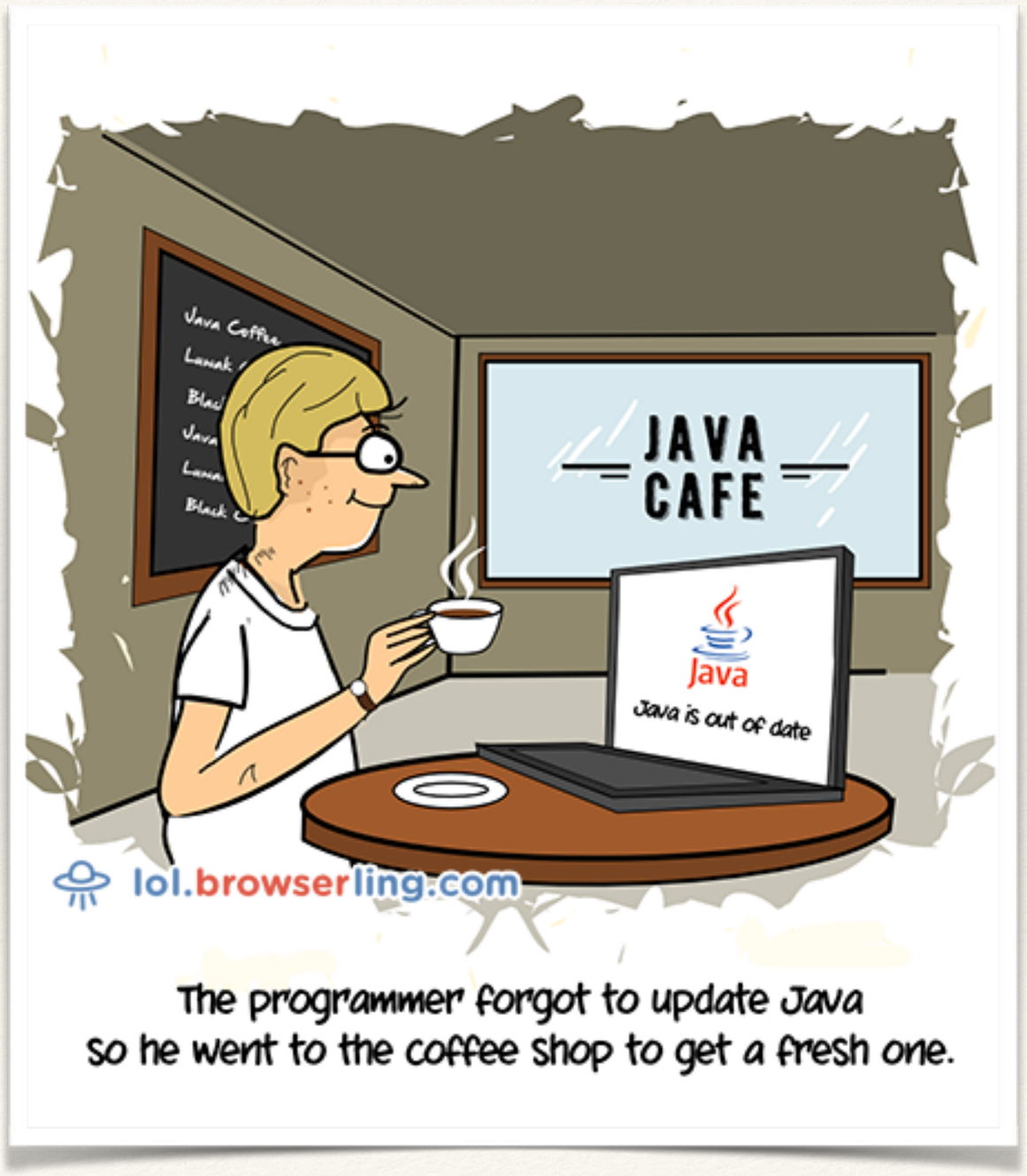
One analyst called such sweeping condemnations of Microsoft an overreaction.

"Microsoft is entitled to extend Java and make it work better on their platform," said Anne Thomas, senior analyst at the Boston-based Patricia Seybold Group, Inc. ■

https://goo.gl/GDlZOS

Java Evolution

- Java 5 (2004):
  - generics(docs)
  - autoboxing/unboxing (docs)
  - for-each loop (docs)
  - annotations, enums, etc.
- Java 7 (2011)
  - diamond type inference (docs)
  - binary literals, String switches, etc.
- Java 8 (2014)
  - lambda expressions (docs)
- Java 9 (2016?)
  - native Java REPL?



Java Refresher (contrasts vs C)

- No preprocessor
  - `import` instead of `#include`
- No `const`: `static final` is effectively similar
- Type sizes are specified, not environment-dependent

Java Refresher (contrasts vs C)

- No preprocessor
  - `import` instead of `#include`
- No `const`: `static final` is effectively similar
- Type sizes are specified, not environment-dependent
- All arrays are dynamic: `int [] a = new int[10];`
- C-like control flow, with a few exceptions

```
public class label {
    public static void main(String[] args) {
        int i = 1;
        outer:
        while(true) {
            int j = i;
            inner:
            while(true) {
                if(j % 2 == 0)
                    if(j % 4 == 0)
                        if(j % 8 == 0)
                            if(j % 16 == 0)
                                break outer;
                            else
                                break inner;
                        else {i++; continue outer;}
                    else {j++; continue inner;}
                System.out.println(i + " " + j);
                j++;
            }
            i++;
        }
        System.out.println("done");
    }
}
```

Java Refresher (contrasts vs C)

- No preprocessor
  - `import` instead of `#include`
- No `const`: `static final` is effectively similar
- Type sizes are specified, not environment-dependent
- All arrays are dynamic: `int [] a = new int[10];`
- C-like control flow, with a few exceptions
- Strings are *almost* a primitive type, and are immutable

Java String Pooling

```
public class Class2 {
  public static String hw = "hello, world!";
}

public class Class1 {
  public static String hw = "hello, world!";
  public static void main(String[] args) {
    String mainS  = "hello, world!";
    String newmS  = new String(mainS);
    String newS   = new String("hello, world!");
    String builtS = "hello" + ", " + "world!";

    System.out.println("Class2.hw: "+("hello, world!" == Class2.hw));
    System.out.println("       hw: "+("hello, world!" == hw));
    System.out.println("    mainS: "+("hello, world!" == mainS));
    System.out.println("    newmS: "+("hello, world!" == newmS));
    System.out.println("     newS: "+("hello, world!" == newS));
    System.out.println("   builtS: "+("hello, world!" == builtS));
  }
}
```

Java String Pooling

```
public class Class2 {
  public static String hw = "hello, world!";
}

public class Class1 {
  public static String hw = "hello, world!";
  public static void main(String[] args) {
    String mainS  = "hello, world!";
    String newmS  = new String(mainS);
    String newS   = new String("hello, world!");
    String builtS = "hello" + ", " + "world!";

    System.out.println("Class2.hw: "+("hello, world!" == Class2.hw));
    System.out.println("      hw: "+("hello, world!" == hw));
    System.out.println("   mainS: "+("hello, world!" == mainS));
    System.out.println("   newmS: "+("hello, world!" == newmS));
    System.out.println("    newS: "+("hello, world!" == newS));
System.out.println("   newSi: "+("hello, world!" == newS.intern()));
    System.out.println("  builtS: "+("hello, world!" == builtS));
  }
}
```

Object Orientation in Java

- Every program a class, every object an `Object`
- References instead of pointers for objects
  - Be *very* wary of ==
- Autoboxing of primitive types into objects
  - Why?

Wrapper Types

- Many libraries only work with objects (e.g. Collections)
- Types exist to "wrap up" primitive types into objects
  - `Boolean`, `Character`, `Double`, `Integer`, etc.
- Wrapper type classes also give some utility fields/methods
  - `Integer.parseInt("123"); // atoi`
  - `Integer.valueOf("123"); // atoI`
  - `Double.POSITIVE_INFINITY`, etc.

Wrapper Types

- Before Java 1.5, explicit conversion between corresponding primitive and wrapper types:
  - `Integer I = new Integer(123);`
  - `int i = I.intValue();`
- Since Java 1.5 autoboxing/autounboxing:
  - `Integer I = 123; // no constructor`
  - `int i = I;  // no extractor`

Autoboxing Pitfalls

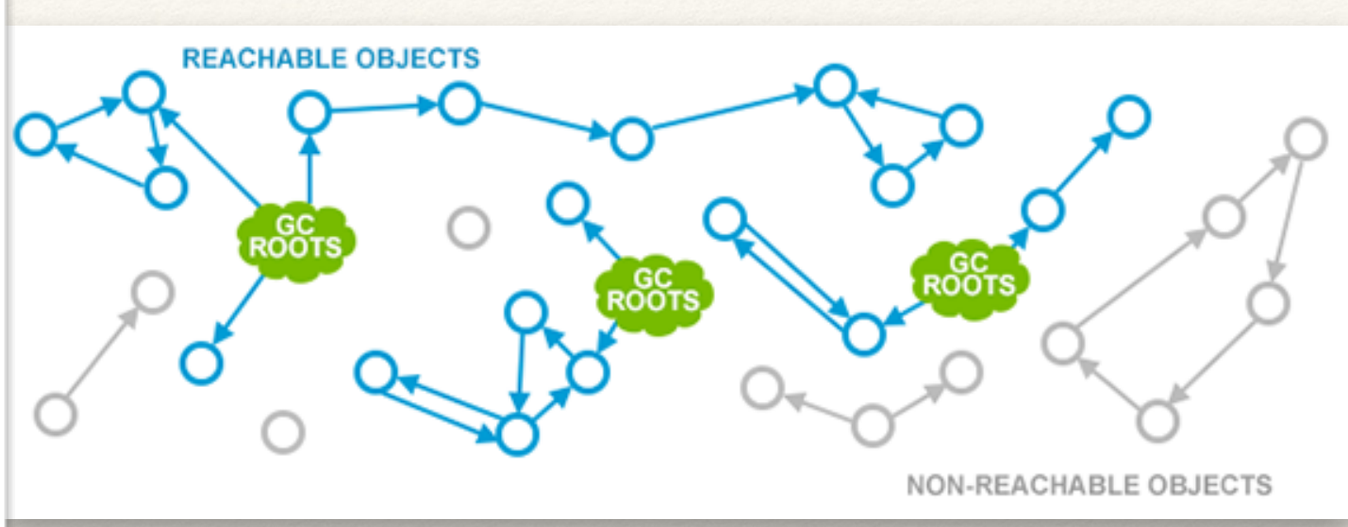
- Example adapted from Josh Bloch —
  What does this print?

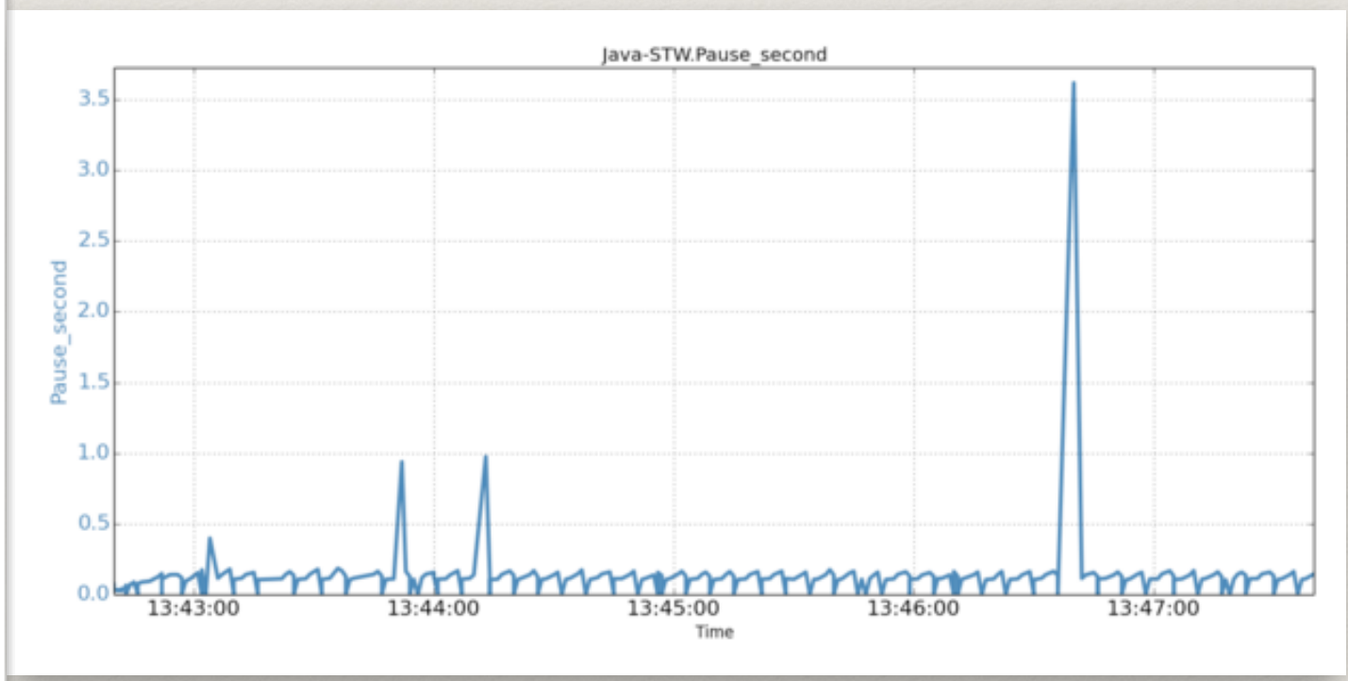
```
public class box {
    public static void main(String[] args) {
        cmp(new Integer(42), new Integer(42));
    }

    static void cmp(Integer first, Integer second) {
        if (first < second)
            System.out.printf("%d < %d\n", first, second);
        else if (first > second)
            System.out.printf("%d > %d\n", first, second);
        else if (first == second)
            System.out.printf("%d == %d\n", first, second);
    }
}
```

Object Orientation in Java

- Every program a class, every object an `Object`
- References instead of pointers for objects
  - Be *very* wary of ==
- Autoboxing of primitive types into objects
- Garbage collection


REACHABLE OBJECTS
GC ROOTS
GC ROOTS
GC ROOTS
NON-REACHABLE OBJECTS

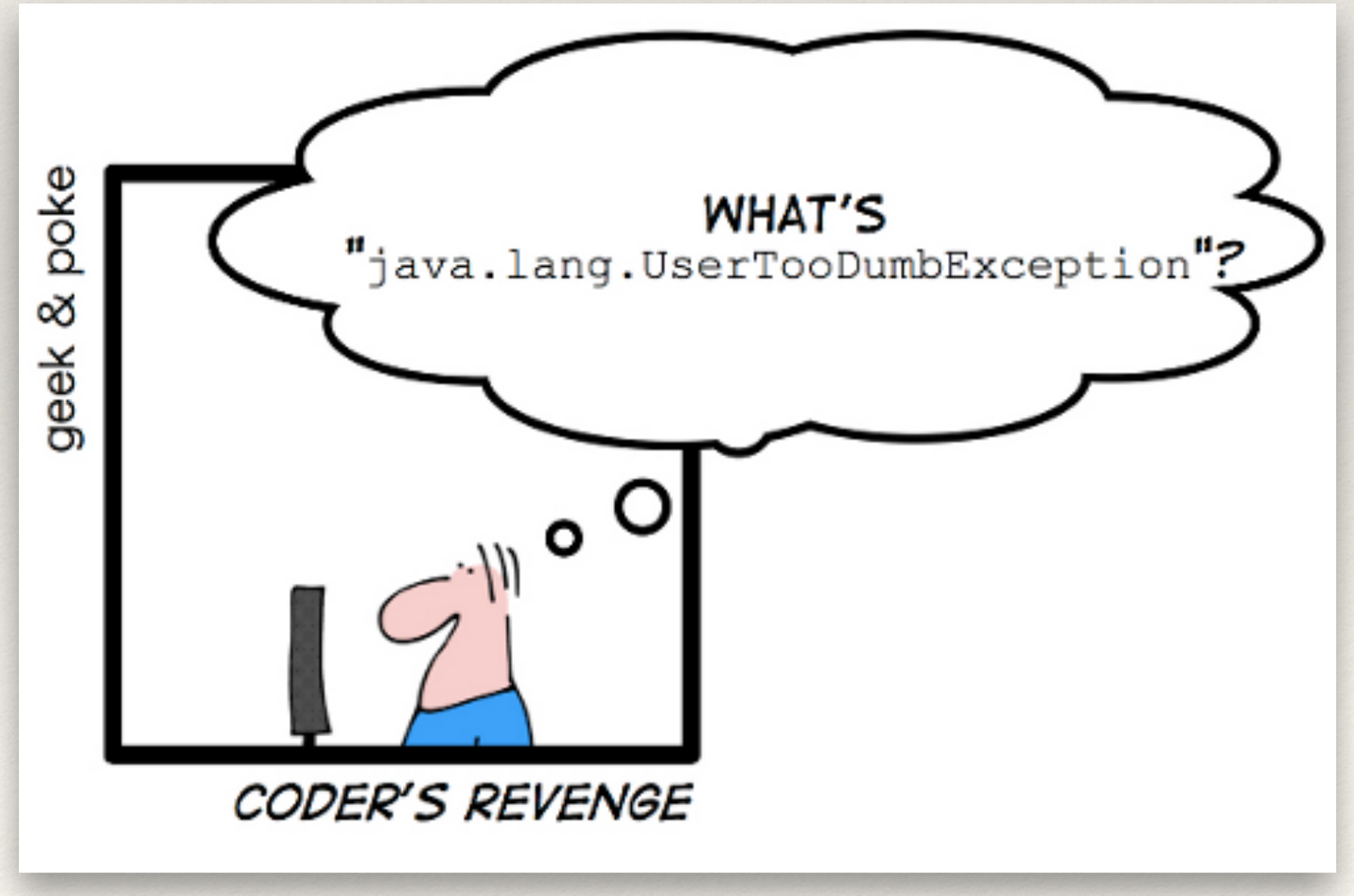

Java-STW.Pause_second
Pause_second
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
13:43:00
13:44:00
13:45:00
13:46:00
13:47:00
Time

Object Orientation in Java

- Every program a class, every object an `Object`
- References instead of pointers for objects
  - Be *very* wary of ==
- Autoboxing of primitive types into objects
- Garbage collection
- Exceptions: `try {…} catch (E) {…} finally {…}`

Java Error Handling

- C "semantics":
  - ignore as if it can't happen
  - return a meaningful value
    - this is pretty terrible …
- Java exceptions:
  - `throw`
  - `catch`
    - or pass up to caller



try, catch, finally

```
public void foo() {
  try {
       // if anything here throws an IO exception
        // or a subclass, like FileNotFoundException
  } catch (IOException e) {
      // this code will be executed to deal with it
  } finally {
      // this is done regardless
  }

// method must list exceptions it can throw, explicitly or not
public void foo2() throws IOException {
   // if anything here throws any kind of IO exception
   // foo will throw an exception, to be handled by its caller
}
```

Rationale for Java Exceptions

- In Java you can ignore exceptions, but you have to willfully do it. You can't accidentally say, "I don't care." You have to explicitly say, "I don't care."

Other Advice on Exceptions

- Do... or do not. There is no try.

* observation stolen from Brian Kernighan

Separated at Birth?

Exceptional Abuse?

```
class Abuse {
  public static void main(String[] args) {
     Abuse e = new Abuse(Integer.parseInt(args[0]));
  }

  public Abuse(int n) {
     System.out.println(f(n));
  }

  private int f(int x) {
     try {
        if (x <= 0)
           return 1;
        else
           throw new Exception();
     } catch (Exception e) {
        return x * f(x-1);
     }
  }
}
```

Object Orientation in Java

- Every program a class, every object an `Object`
- References instead of pointers for objects
  - Be *very* wary of ==
- Autoboxing of primitive types into objects
- Garbage collection
- Exceptions: `try {…} catch (E) {…} finally {…}`
- Polymorphism

Polymorphism in Java

- An object that "IS-A" to multiple classes is polymorphic
  - Trivially, all objects are polymorphic with `Object`
  - Reference variables can refer to any object of their type, ***or any subtype*** of their type!

So far so good …

```
class Aircraft {
    public String abilities() { return "can fly"; }
}

class Helicopter extends Aircraft {
    public String abilities() { return "can fly and hover";}
}

public class TestPoly {
    public static void main(String[] args) {
        Aircraft a = new Aircraft();
        Helicopter h = new Helicopter();
        System.out.println(a.abilities());
        System.out.println(h.abilities());
        Aircraft h2 = new Helicopter();
        System.out.println(h2.abilities());
    }
}
```

Polymorphism in Java

- An object that "IS-A" to multiple classes is polymorphic
  - Trivially, all objects are polymorphic with `Object`
  - Reference variables can refer to any object of their type, ***or any subtype*** of their type!
  - But when referring to objects of their subtype, they're still limited by their own type …

Careful …

```
class Aircraft {
    public String abilities() { return "can fly"; }
}

class Helicopter extends Aircraft {
    public String abilities() { return "can fly and hover";}
    public int getNumBlades() { return 2; }
}
```

```
public class TestPoly {
    public static void main(String[] args) {
        Aircraft a = new Aircraft();
        Helicopter h = new Helicopter();
        System.out.println(a.abilities());
        System.out.println(h.abilities());
        Aircraft h2 = new Helicopter();
        System.out.println(h2.abilities());

        System.out.println(h.getNumBlades());
        System.out.println(h2.getNumBlades()); // doesn't compile
    }
}
```

```
TestPoly.java:21: error: cannot find symbol
            System.out.println(h2.getNumBlades());
                               ^
  symbol:   method getNumBlades()
```

Object Orientation in Java

- Every program a class, every object an `Object`
- References instead of pointers for objects
  - Be *very* wary of ==
- Autoboxing of primitive types into objects
- Garbage collection
- Exceptions: `try {…} catch (E) {…} finally {…}`
- Polymorphism
- Interfaces and Object hierarchies

Interfaces

- An interface is like a class, but without implementation
  - Only declares method prototypes, not bodies
    - implicitly `public`
  - Only declares constant fields
    - implicitly `public static final`
- Any class can `implement` the interface

Java 7 Diamond Operator

Canonical example from Jeremy Manson:

```
Map<String, List<String>> myMap = new HashMap<String, List<String>>();
```

```
Map<String, List<String>> myMap = new HashMap< >();
```