Administrivia

• Project 1 posted today
• Java readings from Thinking in Java
  – we’ll do parts of many chapters, but definitely read Chapter 1 for an overview
  – I’ll add some suggestions on Web page, but use it as a reference
Outline

- Object oriented programming principles
  - How Java realizes them
  - How Java differs from C++
- Useful information on Java (not covered in class)
  - Using the compiler
  - I/O libraries
  - Container classes

Software Engineering Goals

- Reliability (it works!)
- Performance
- Reusability (write-once, then reuse)
- Maintainability
  - Easy to modify/extend
  - Easy to understand
- Quick development time
Object Orientation

• Abstraction
  – focus on essential properties, ignore unimportant details
• Encapsulation
  – separate external, visible behavior from internal, hidden behavior

Object Orientation (cont’d)

• Combining data and behavior
  – objects, not developers, decide how to carry out operations
• Sharing
  – similar operations and structures are implemented once
• Emphasis on object-structure rather than procedure structure
  – behavior more stable than implementation
  – … but procedure structure still useful
And one more …

- Security and Reliability
  - Write code that works, and is not insecure

Java

- Similar to C++, but with “unsafe” features removed, and others added
- Fully specified, compiles to virtual machine
  - machine-independent
- Secure
  - bytecode verification (“type-safe”)
  - security manager
Java Design

• Everything inherits from **Object***
  – Allows sharing, generics, and more

```java
class Number
    extends Object
interface Comparable
   public int compareTo(Object o)
```

* Well, almost: there are primitive int, long, float, etc.

Java Design

• *Inheritance*
  – Hierarchical code sharing (“is-a”)

• *Interfaces*
  – For “mixins” & non-hierarchical frameworks

```java
class Number
    implements Comparable
```

inherited from

CMSC 433, Fall 2002 - Michael Hicks
Java Design

• Security and reliability
  – Strong type system
    • Object o = (Object)27; not allowed!
  – Garbage collection
    • No free()
  – Exceptions
    • Separation of error-handling from algorithm

Java libraries and features

• Utilities
  – collection classes, Zip files, internationalization
• GUIs, graphics and media
• Networking
  – sockets, URLs, RMI, CORBA
• Threads
• Databases
• Cryptography/security
Some of what’s missing from C++

- Preprocessor (#include, #define, …)
- Some “low-level” types
  - structs and unions
  - enumerated types
  - bit-fields
- Some function features
  - variable-length argument lists
  - operator overloading
- Some class features
  - multiple inheritance (of implementation)
  - templates/ parameterized types (but now in 1.4!)

Naming conventions

- Classes/Interfaces start with a capital letter
  - Object, Number, Thread, …
- packages/methods/variables start lowercase
  - Thread myThread = new Thread();
  - java.lang, org.xml.sax
- Capitalize multi-word names (no underscores)
  - SortedList, compareTo, toBinaryString
- CONSTANTS all in uppercase (use underscores)
  - PI, E, MAX_VALUE
Object-oriented programming in Java

Java Classes and Objects

- Each object is an instance of a class
  - an array is an object
- Each class extends one superclass
  - Object if not specified
  - class Object has no superclass
Objects have methods

- All objects, therefore, inherit them
  - Default implementations may not be the ones you want

  public boolean equals (Object that)
  public String toString()
  public int hashCode()
  public void finalize()

  - And others …

Objects and references

- All objects allocated on the heap with new()
  - All variables of non-primitive type are references to an object or null; assignment (=) copies references
  - No object can “contain” another object
  - No objects stack-allocated (only references there)

- Reference is like a C++ pointer, except
  - can only point to start of heap-allocated object
  - no pointer arithmetic allowed
  - use . instead of -> to access fields/methods
String example

```java
String a = "Tri-ple";
String b = a.substring(2, 4);
```

Equality

- **Object** .equals() method
  - Structural ("conceptual") equality
- `==` operator
  - true if arguments reference the same object
  - `o == p` $\rightarrow$ `o.equals(p)`
class Complex – a toy example

```java
public class Complex {
    private double r, i;
    public Complex(double r, double i) {
        this.r = r;
        this.i = i;
    }
    public String toString() {
        return "(" + r + "," + i + ")";
    }
    public Complex plus(Complex that) {
        return new Complex(
            r + that.r,
            i + that.i);
    }
}
```

Using Complex

```java
public static void main(String[] args) {
    Complex a = new Complex(5.5, 9.2);
    Complex b = new Complex(2.3, -5.1);
    Complex c, d;
    c = a.plus(b);
    d = a.plus(b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("c.equals(d): " + (c.equals(d)));
    System.out.println("c == d: " + (c == d));
}
```

prints:
```
a = (5.5,9.2)
b = (2.3,-5.1)
c = (7.8,4.1)
c.equals(d): false
```

CMCS 433, Fall 2002 - Michael Hicks 21

CMCS 433, Fall 2002 - Michael Hicks 22
**Adding equals to Complex**

```java
public class Complex {
    ...
    public boolean equals(Object o) {
        if (o instanceof Complex) {
            Complex c = (Complex)o;
            return (c.r == this.r && c.i == this.i);
        } else {
            return false;
        }
    }
}
```

Runtime test to determine the object's actual class

**Using Complex again**

```java
public static void main(String[] args) {
    Complex a = new Complex(5.5,9.2);
    Complex b = new Complex(2.3,-5.1);
    Complex c,d;
    c = a.plus(b);
    d = a.plus(b);
    System.out.println("a = " + a);
    System.out.println("b = " + b);
    System.out.println("c = " + c);
    System.out.println("c.equals(d): " + (c.equals(d)));
    System.out.println("c == d: "+(c==d));
}
```

prints:

```
a = (5.5,9.2)
b = (2.3,-5.1)
c = (7.8,4.1)
c.equals(d): true
c == d: false
```
Downcasting

• **(Bar) foo**
  – run-time exception if object reference by **foo** is not a subclass of **Bar**
  – compile-time error if **Bar** is not a subtype of **foo** (i.e. it always throws an exception)
  – doesn’t transform anything, just allows treating the result as if it were of type **Bar**

• **o instanceof Foo** returns true iff **o** is an instance of a subclass of **Foo**

Different from C++

• No malloc()
  – Only new()

• No free()
  – Uses garbage collection

• No pointer operations: *, &, ->, +, ++, etc.
  – Simplifies usage and implementation

• Method parameters pass-by-value
  – but object parameters are references to heap objects that can be changed