The JOOS Project

Matthew Dwyer
324E Nichols Hall
dwyer@cis.ksu.edu
http://www.cis.ksu.edu/~dwyer

CIS 706 Translators I

Course Project:

- Java's Object-Oriented Subset;
- compiled into Java Virtual Machine code;
- illustrates a general-purpose language;
- shows client-side programming on the WWW;
- used to teach by example;
- the $$A^-$$ source code will be studied;
- and you will upgrade it into an $$A^+$$ version.
- $$A^-$$ source code is available in C or Java.
Java

- Originally developed as a small, clean, OO language for programming consumer devices;
- Grew dramatically as a web-services language (applets, EJB);
- Now considered a full-featured general purpose language.

Java Compilation

.java → .class

- Java programs are developed as a collection of Java classes;
- each class is compiled into Java Virtual Machine (JVM) bytecode;
- bytecode is interpreted using some implementation of the JVM; and
- many browsers support the interpretation of JVM bytecode.
Benefits of Java

• it's object-oriented;
• it's a ``cleaner'' OO language than \( \text{C++} \);
• it's portable (except for native classes);
• it's distributed and multithreaded;
• it's secure; and
• it supports windowing and applets.

Java Security

• programs are strongly type-checked at compile-time;
• array bounds are checked at run-time;
• null pointers are checked at run-time;
• there are no explicit pointers;
• dynamic linking is checked at run-time; and
• class files are verified at load-time.
JOOS Goals

- extract the essence of the object-oriented subset of Java;
- make the language small enough for a course project, yet
- large enough to be interesting;
- provide a mechanism to link to existing Java code; and
- ensure that every JOOS program is a valid Java program.

Programming in JOOS

- each JOOS program is a collection of classes;
- there are ordinary classes which are used to develop JOOS code; and
- there are external classes which are used to interface to Java libraries.
- An ordinary class consists of:
  - protected fields;
  - exactly one constructor; and
  - public methods.
Example

```java
public class Cons {
  protected Object first;
  protected Cons rest;

  public Cons(Object f, Cons r) {
    super();
    first = f; rest = r;
  }

  public void setFirst(Object newfirst) {
    first = newfirst;
  }

  public Object getFirst() { return first; }

  public Cons getRest() { return rest; }

  public boolean member(Object item) {
    if (first.equals(item))
      return true;
    else if (rest == null)
      return false;
    else
      return rest.member(item);
  }

  public String toString() {
    if (rest == null)
      return first.toString();
    else
      return first + " " + rest;
  }
}
```

Example: Notes

- fields in JOOS must be `protected`
  - they can only be accessed via objects of the class or its subclasses;
- constructors in JOOS must be unique and start by invoking the constructor of the superclass;
- methods in JOOS must be `public`
  - they can be invoked by any object; and
- constructors and methods in JOOS may not be overloaded.
Inheritance

• Class hierarchies are single inheritance
  – Each class has one superclass, except for the root
• The root class is called Object
  – Any class without an explicit extends clause is a subclass of Object

```java
public class Cons { … }
public class Cons extends Object { … }
```

Object methods

```java
toString( )
    returns a string encoding the type and object id; and
equals( Object obj )
    returns true if the object reference denotes the current object.
```

These methods are often overridden in subclasses:
• toString encodes the value as a string; and
• equals decides a more abstract equality.
• When overriding a method, the argument and return types must remain the same
Extending Cons

```java
public class ExtCons extends Cons {
    protected int intField;

    public ExtCons(Object f, Cons r, int i) {
        super(f, r);
        intField = i;
    }

    public void setIntField(int i) {
        intField = i;
    }

    public int getIntField() {
        return intField;
    }
}
```

Using Cons

```java
import joos.lib.*;
public class UseCons {
    public UseCons() { super(); }
    public static void main(String argv[]) {
        Cons l;
        JoosIO f;
        l = new Cons("a", new Cons("b", new Cons("c", null)));
        f = new JoosIO();
        f.println(l.toString());
        f.println("first is ", l.getFirst());
        f.println("second is ", l.getRest().getFirst());
        f.println("a member? ", l.member("a"));
        f.println("z member? ", l.member("z"));
    }
}
```
Running Program

Output
a b c
first is a
second is b
a member? true
z member? false

JOOS Types

• Types in JOOS are either primitive types:
  – boolean: true and false
  – int: -2^31 … 2^31-1
• or user-defined class types;
• or externally defined class types:
  – Object, Boolean, Integer, String
  – BitSet, Vector, Date, ...
Properties of JOOS Types

• JOOS is strongly-typed;
• JOOS uses the name of a class as its type;
• given a type of class C any instance of class C or a subclass of C is a permitted value;
• there is `down-casting` which is automatically checked at run-time;
• there is an explicit `instance_of` check; and
• some type-checking must be done at run-time.

JOOS Statements

• Expression Statements
  \[ x = y + z; \]
  \[ x = y = z; \]
  Cons.toString(l);
  new Cons("abc", null);
• Block Statements
  \{ int x;
    x = 3;
  \}
JOOS Statements

- **Control Structures**
  
  ```java
  if (l.member("z")) {
    // do something
  }
  ```

  ```java
  while (l != null) {
    // do something
    l = l.getRest();
  }
  ```

- **Return Statements**
  
  ```java
  return;
  ```

  ```java
  return true;
  ```

JOOS Expressions

- **Constant Expressions**
  
  ```java
  true, 13, "abc", null
  ```

- **Variable Expressions**
  
  ```java
  i, first, rest
  ```

- **Binary Expressions**
  
  ```java
  ||, &&, !=, ==, <, >, <=, >=, instanceof, +, -, *, /, %
  ```

- **Unary Expressions**
  
  ```java
  -, !
  ```
JOOS Expressions

- Instance Creation
  \[\text{new } \text{Cons(“abc”, nul l)}\]

- Cast Expressions
  \[(\text{String}) \text{getFirst(list)}\]

- Method Invocation
  \[l \text{.getFirst()}
  \text{super.getFirst();}
  l \text{.getFirst().getFirst();}
  \text{this.get_first();}\]

Abstract Methods and Classes

- A method may be \textit{abstract}, where no implementation is given;
- If a class contains one or more \textit{abstract} methods, it must be defined as an \textit{abstract} class;
- The constructor of an \textit{abstract} class cannot be invoked;
- \textit{abstract} classes are used to define "frameworks".
A Framework

```java
import joos.lib.*;

public abstract class Benchmark {
    // JOOS interface to system class
    protected JoosSystem s;

    public Benchmark() {
        super();
        s = new JoosSystem();
    }

    public abstract void benchmark();
    // hook for actual benchmark
}
```

Framework Use

```java
import joos.lib.*;

public class UseBenchmark {
    public UseBenchmark() {
        super();
    }

    public static void main(String argv[]) {
        ExtBenchmark b;
        JoosIO f;
        int reps, time;
        b = new ExtBenchmark();
        f = new JoosIO();
        f.print("Enter number of repetitions: ");
        reps = f.readInt();
        time = b.myrepeat(reps);
        fprintln("time is " + time + " millisecs");
    }
```

Final Methods and Classes

- a final method cannot be overridden by subclasses;
- it is used when no modification in the functionality is allowed;
- a final class cannot be extended;
- all methods in a final class are assumed to be final; and
- final classes are typically libraries: Boolean, Integer, and String.

Synchronized Methods

- JOOS programs can start multiple threads;
- sometimes a resource must be locked, so multiple threads can't access it at the same time;
- JOOS provides synchronized methods; and
- when a thread invokes a synchronized method on an object, the object is locked until the method completes.
Synchronized Example

```java
public class SyncBox {
    protected Object boxContents;
    public SyncBox() { super(); }
    public synchronized Object get() {
        Object contents;
        contents = boxContents;
        boxContents = null;
        return contents;
    }
    public synchronized boolean put(Object contents) {
        if (boxContents != null) return false;
        boxContents = contents;
        return true;
    }
}
```

External Classes

- **in Java**
  - Java compiles programs with respect to a set of libraries of precompiled class files; and
  - when a Java compiler encounters an unknown method, it searches the precompiled bytecode for an implementation.
- **in JOOS**
  - JOOS compiles programs with respect to a set of libraries of precompiled class files; but
  - external classes must be explicitly presented to the JOOS compiler.
JOOS External Example

```java
extern public final class String in "java.lang" {
    public String(String value);
    public String toString();
    public boolean equals(Object obj);
    public int length();
    public boolean equalsIgnoreCase(String anotherString);
    public int compareTo(String anotherString);
    public boolean startsWith(String prefix, int toffset);
    public boolean endsWith(String suffix);
    ...
    public String concat(String str);
    public String toLowerCase();
    public String toUpperCase();
    public String trim();
}
```

External Libraries

- External declarations Java libraries
  - javalib.joos
  - appletlib.joos
  - awtlib.joos
  - netlib.joos
- External declarations for JOOS libraries
  - jooslib.joos
Example JOOS Programs

- AppletGraphics
- AwtDemos
- ImageDemos
- Network
- Simple
- Threads
- WIGapplets

JOOS versus Java

- does not support packages, interfaces, exceptions, some control structures, mixed statements and declarations;
- has only protected fields and public methods;
- does not allow overloading;
- does not support arrays;
- does not allow static functions;
- supports only int and boolean for basic types; and
- uses external class declarations.