# The JOOS language

The Java language:

- was originally called Oak;
- was developed as a small, clean, OO language for programming consumer devices;
- was built into the Webrunner browser;
- matured into Java and HotJava; and
- is now supported by many browsers, allowing Java programs to be embedded in WWW pages.

Basic compilation (`.java` $\longrightarrow$ `.class`):

- Java programs are developed as a collection of Java classes;
- each class is compiled into Java Virtual Machine (JVM) bytecode;
- bytecode is interpreted or (JIT) compiled using some implementation of the JVM;
- Java supports GUI; and
- many browsers has java plugins for executing JVM bytecode.

Major benefits of Java:

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

Java security has many sources:

- 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.

Major drawbacks of Java:

- it's slow; zzzzz
- it misses some language features, e.g. genericity;
- it does not have a standard yet (`JDK 1.0.2` vs. `JDK 1.1.*` vs. . . . ); and
- it's not JOOS.

Goals in the design of JOOS:

- 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:

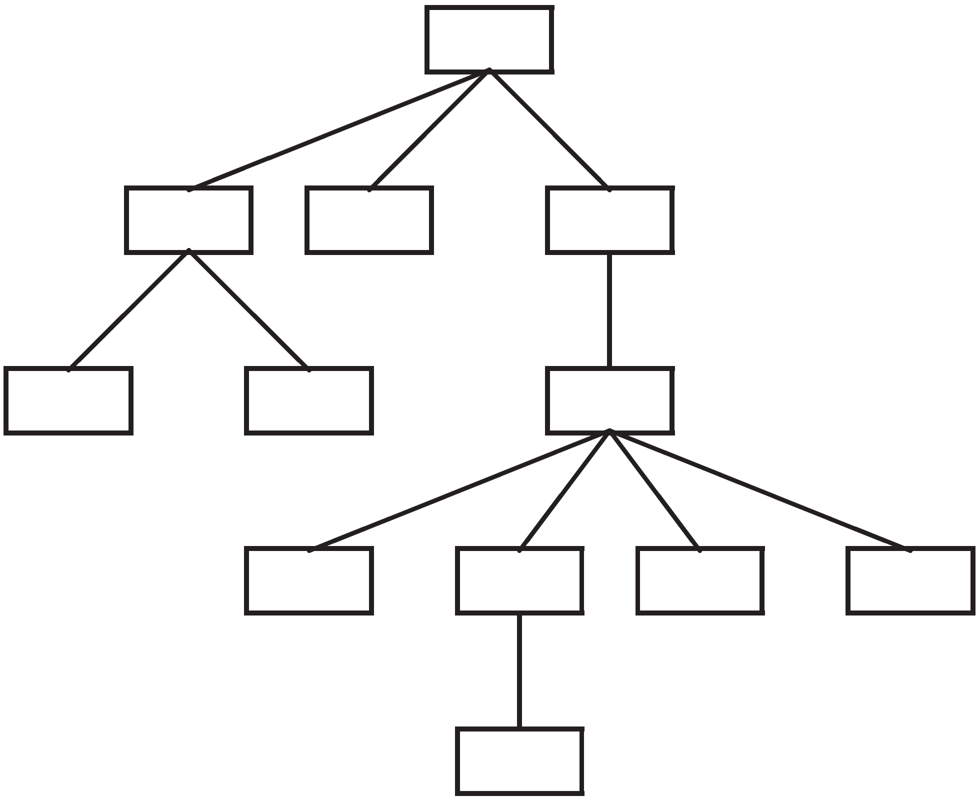
- protected fields;
- constructors; and
- public methods.

```
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;
  }
}
```

Notes on the `Cons` example:

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

The class hierarchies in JOOS and Java are both single inheritance, i.e. each class has exactly one superclass, except for the root class:



The root class is called `Object`, and any class without an explicit extension is a subclass of `Object`.

The definition of `Cons` is equivalent to:

```
public class Cons extends Object
{ ... }
```

which gives the tiny hierarchy:

```
Object
public String toString();
public boolean equals(Object obj);
```

```
Cons
public void setFirst(Object newfirst);
public Object getFirst();
public Cons getRest();
public boolean member(Object item);
public String toString();
```

The class `Object` has two methods:

- `toString` returns a string encoding the type and object id; and
- `equals` returns true if the object reference denotes the current object.

These methods are often overridden in subclasses:

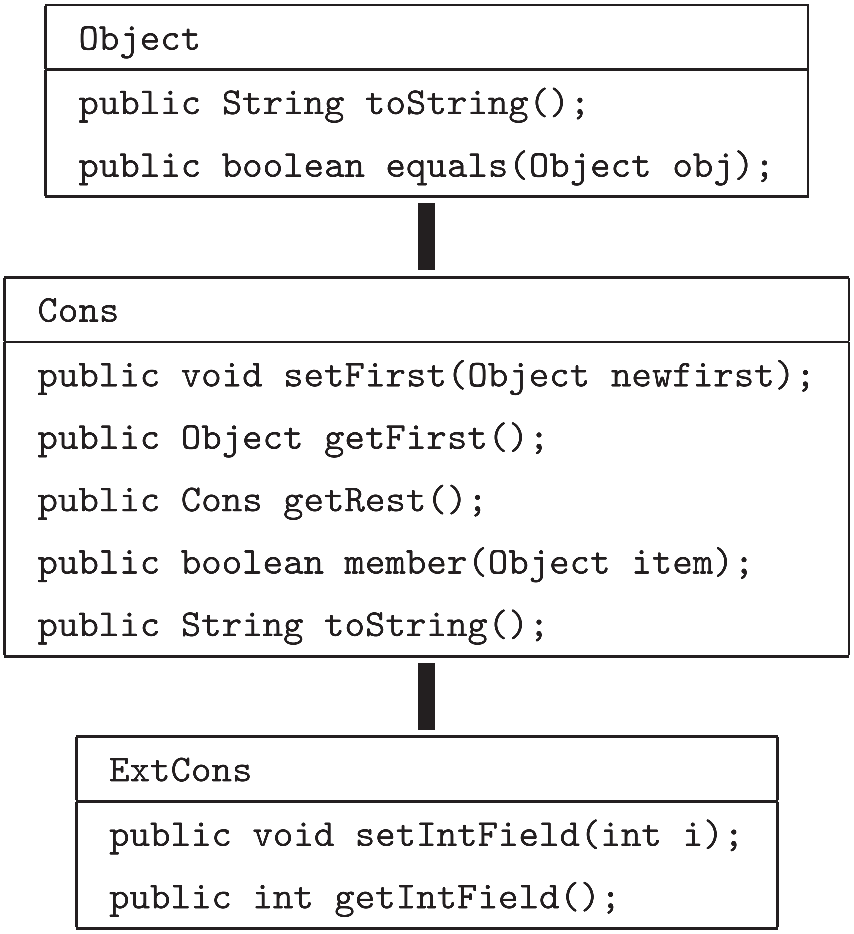
- `toString` encodes the value as a string; and
- `equals` decides a more abstract equality.

When overriding a method, the argument and return types must remain the same.

# Extending the `Cons` class:

```
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); }
}
```

The extended hierarchy:



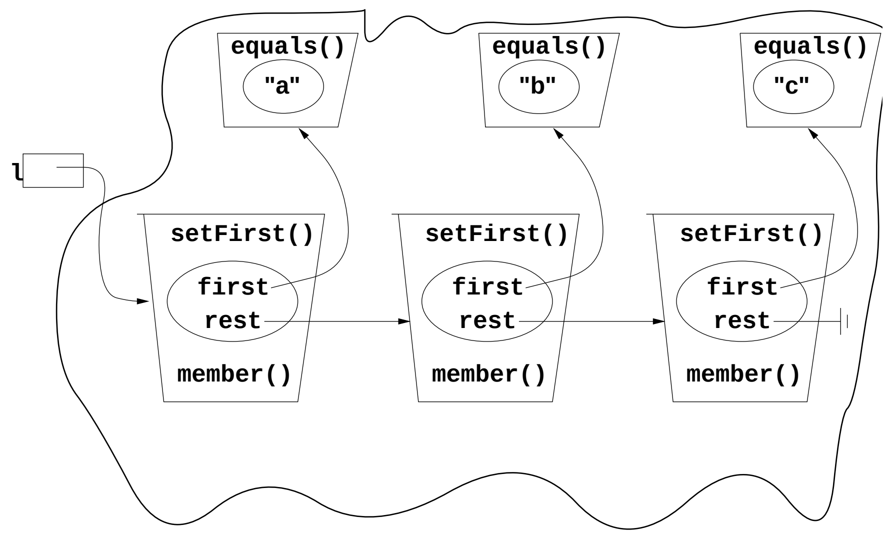
## Using the `Cons` class:

```
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"));
    }
}
```

A Java program (not an applet) requires a `main` method.

It is necessary to `import` library functions such as `println`.

The `UseCons` program builds these objects:




The output of the program is:

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

Types in JOOS are either primitive types:

- `boolean`: `true` and `false`;
- `int`: $-2^{31} \ldots 2^{31} - 1$;
- `char`: the ASCII characters;

or user-defined class types;

or externally defined class types:

- `Object`;
- `Boolean`;
- `Integer`;
- `Character`;
- `String`;
- `BitSet`;
- `Vector`;
- `Date`.

Note that `boolean` and `Boolean` are different.

Types in Java and JOOS:

- Java is strongly-typed;
- Java 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.

## Statements in JOOS:

- expression statements:

```
x = y + z;
x = y = z;
a.toString(l);
new Cons("abc",null);
```

- block statements:

```
{ int x;
  x = 3;
}
```

- control structures:

```
if (l.member("z")) {
   // do something
};

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

- return statements:

```
return;

return true;
```

# Expressions in JOOS:

- constant expressions:

  `true, 13, '\n', "abc", null`

- variable expressions:

  `i, first, rest`

- binary expressions:

  ```
  ||
  &&
  != ==
  < > <= >= instanceof
  + -
  * / %
  ```

- unary expressions:

  ```
  -
  !
  ```

## Expressions in JOOS:

- class instance creation:

```
new Cons("abc",null)
```

- cast expressions:

```
(String) getFirst(list)
(char)119
```

- method invocation:

```
l.getFirst()
super.getFirst();
l.getFirst().getFirst();
this.getFirst();
```

Abstract methods and classes:

- a method may be `abstract`, where no implementation is given;
- if a class contains one or more `abstract` methods, it must be defined as an `abstract` class;
- the constructor of an `abstract` class cannot be invoked;
- `abstract` classes are used to define "frameworks".

```
import joos.lib.*;

public abstract class Benchmark {

  protected JoosSystem s;  // JOOS interface to
                           // the Java System Class

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

  // Hook for actual benchmark
  public abstract void benchmark();

  // driver to time repeated executions
  public int myrepeat(int count)
  { int start;
    int i;

    start = s.currentTimeMillis();
    i = 0;
    while (i < count) {
      this.benchmark();
      i = i+1;
    }
    return s.currentTimeMillis()-start;
  }
}
```

```
import joos.lib.*;

public class ExtBenchmark extends Benchmark {
  public ExtBenchmark() {
    super();
  }

  public void benchmark() {}   // timing an empty method
}

public class UseBenchmark {

  public UseBenchmark() { super(); }

  public static void main(String argv[])
  { ExtBenchmark b;
    JoosIO f;
    int reps;
    int time;

    b = new ExtBenchmark();
    f = new JoosIO();

    f.print("Enter number of repetitions: ");
    reps = f.readInt();
    time = b.myrepeat(reps);
    f.println("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:

- Java and JOOS programs can start multiple threads;
- sometimes accessing a resource must be in a critical section, so only one thread can be in the critial section;
- each object has a lock and JOOS provides `synchronized` methods; and
- when a thread invokes a `synchronized` method on an object, the thread acquires the lock until the method completes.

```
public class SyncBox {
  protected Object boxContents;

  public SyncBox() { super(); }

  // return contents of the box, set contents to null
  public synchronized bject get()
  {
    Object contents;
    contents = boxContents;
    boxContents = null;
    return contents;
  }

  // put something in the box,
  // if the box already has something in it, return false
  // else fill the box, return true
  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.

External classes 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;

```
// java.lang.String
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 int indexOf(String str, int fromIndex);
  public int lastIndexOf(String str, int fromIndex);
  public boolean regionMatches(
                  boolean ignoreCase, int toffset,
                  String other, int ooffset, int len);
  public String substring(int beginIndex, int endIndex);
  public String concat(String str);
  public String toLowerCase();
  public String toUpperCase();
  public String trim();
}
```

External declarations for Java libraries:

- `javalib.joos`
- `appletlib.joos`
- `awtlib.joos`
- `netlib.joos`

External declarations for JOOS libraries:

- `jooslib.joos`

Example JOOS programs:

- `AppletGraphics`: simple graphics programs to be displayed via a browser;
- `AwtDemos`: examples of using the Abstract Windows Toolkit;
- `ImageDemos`: two techniques for displaying an animation;
- `Network`: simple examples of interacting over the network;
- `Simple`: a relatively large collection of simple Java programs;
- `Threads`: simple, multi-thread programs; and
- `WIGapplets`: examples of WIG applets.

JOOS compared to 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 of methods;
- does not support arrays;
- does not allow static functions;
- supports only `int`, `boolean`, and `char` for basic types; and
- uses external class declarations.