Lecture 8

Object-oriented programming (Sethi Ch 6 and 7)

What is OOP?

Examples in C++ and Java
Admin

Discussion of homework sheet 6 and 7

Any questions?
Object-oriented programming

What is OOP?

According to Sethi:

- Procedures
- Modules
- Abstract Data Types
- Inheritance

According to Eckel (*Thinking in C++*):

- Objects = characteristics + behaviors
- Inheritance
- Polymorphism
Evolution of C++ programmers

According to Eckel:

- A better C
- Object-based programming
- Object-oriented programming

Object-based programming = using classes and objects.

Object-oriented programming = object-based + using virtual methods / polymorphism.
Example in C++: abstract class

class Stack {
    public:
        virtual void push (int) = 0;
        virtual int pop () = 0;
        virtual bool isFull () = 0;
        virtual bool isEmpty () = 0;
        virtual Stack* copy () = 0;
};
Example: concrete subclass

class FixedStack : public Stack {
    private:
        int *contents;
        int top;
        int max;

    public:
        FixedStack (int max) {
            this->contents = new int[max];
            this->top = 0;
            this->max = max;
        }

        virtual void push (int x) { contents[top++] = x; }
        virtual int pop () { return contents[--top]; }
        virtual bool isFull () { return top == max; }
        virtual bool isEmpty () { return top == 0; }

        virtual Stack* copy () {
            FixedStack *result = new FixedStack (max);
            for (int i=0; i<top; i++) {
                result->contents[i] = contents[i];
            }
            result->top = top;
            return result;
        }
};
Example: putting it all together

main () {
    Stack *s = new FixedStack (5);
    s->push (1);
    s->push (2);
    s->push (3);
    cout << "Popping: " << s->pop();
    cout << ", " << s->pop () ;
    cout << ", " << s->pop () << endl;
}

The source code is on-line.
Example in Java: abstract class

abstract public class Stack {
    abstract public void push (int x);
    abstract public int pop ();
    abstract public boolean isFull ();
    abstract public boolean isEmpty ();
    abstract public Stack copy ();
}

Stack.java is on-line.
Example: concrete subclass

public class FixedStack extends Stack {

    private int[] contents;
    private int top;
    private int max;

    public FixedStack (int max) {
        this.contents = new int[max];
        this.top = 0;
        this.max = max;
    }

    public void push (int x) { contents[top++] = x; }
    public int pop () { return contents[--top]; }
    public boolean isFull () { return top == max; }
    public boolean isEmpty () { return top == 0; }

    public Stack copy () {
        FixedStack result = new FixedStack (max);
        for (int i=0; i<max; i++) {
            result.contents[i] = contents[i];
        }
        result.top = top;
        return result;
    }
}

FixedStack.java is on-line.
Example: putting it all together

```java
public class FixedStackDemo {
    public static void main (String[] args) {
        Stack s = new FixedStack (5);
        s.push (1);
        s.push (2);
        s.push (3);
        System.out.println ("Popping: ", s.pop() + ", ", s.pop() + ", ", s.pop() + ");
    }
}

FixedStackDemo.java is on-line.
```
Object-orientation

What’s going on here?

- **Encapsulation**: the FixedStack class contains both the data and the code for array-based stacks.
- **Inheritance**: the FixedStack class is a subclass of Stack.
- **Polymorphism**: classes can use Stack objects without knowing the underlying implementation.

In C++ the key to this is: *heap-allocated classes* and *virtual methods*.

In Java, all objects are heap-allocated, and (almost) all methods are virtual.
What is an object?

An object is an instance of a class.

It contains data (fields) and code (methods).

New objects are created with:

    new Bar (...parameters...);

for example:

    Stack *s = new FixedStack (5);  // C++
    Stack s = new FixedStack (5);    // Java
What can you do with an object?

Call its methods:

```cpp
s->push (5);    // C++
s.push (5);     // Java
```

Access its fields (if you’re allowed to):

```cpp
int x = s->top; // C++
int x = s.top;  // Java
```

Both of these will fail because s does not have a public top field.

It’s good OO practice to hide all fields, and only allow accessor methods.
**Accessor methods**

**Bad practice:**

```cpp
class FixedStack : public Stack {
    public:
        int *contents;
        int top;
        int max;
    ...
}
```

**Good practice:**

```cpp
class FixedStack : public Stack {
    private: // or protected:
        int *contents;
        int top;
        int max;
    ...
}
```

Forces programmers to use the API, not the underlying implementation (which might change)!
Static vs Dynamic method dispatch

When a method is called, which code gets run?

\[ s->\text{push}(1); \]
\[ \text{cout} \ll s->\text{pop}(); \]

At compile-time: \( s \) has type Stack.

At run-time: \( s \) has type FixedStack.

Two possibilities:

- **Static dispatch**: call the method of the compile-time type (e.g. \( \text{Stack}::\text{push}(1) \)).
- **Dynamic dispatch**: call the method of the run-time type (e.g. \( \text{FixedStack}::\text{push}(1) \)).

In C++:

- *virtual methods on heap-allocated objects use dynamic dispatch,*
- *non-virtual methods or stack-allocated objects use static dispatch.*

In this case we want dynamic dispatch, so \text{push} and \text{pop} are virtual, and \( s \) is a heap-allocated object.

In Java, (almost) all method dispatch is dynamic.
Static vs Dynamic method dispatch example

What does this do?

class A {
    public:
    void foo () { cout << "A::foo" << endl; }
    virtual void bar () { cout << "A::bar" << endl; }
};
class B : public A {
    public:
    void foo () { cout << "B::foo" << endl; }
    virtual void bar () { cout << "B::bar" << endl; }
};
main () {
    A *heapA = new A ();
    A *heapB = new B ();
    A stackA = A ();
    A stackB = B ();
    heapA->foo ();
    heapA->bar ();
    heapB->foo ();
    heapB->bar ();
    stackA.foo ();
    stackA.bar ();
    stackB.foo ();
    stackB.bar ();
}

Source and answer are on-line.
Real example: Linked lists

An abstract linked list class:

class List {
    public:
        static List* nil ();
        virtual List* cons (int x);
        virtual int head () = 0;
        virtual List* tail () = 0;
        virtual bool isNil () = 0;
    };

A bit later on, we can define:

List* List::nil () {
    static List* result = new NilList ();
    return result;
}
List* List::cons (int x) {
    return new ConsList (x, this);
}

but we have to define NilList and ConsList first.
Real example: Linked lists

A concrete class for the empty list:

class NilList : public List {
    public:
        virtual int head () { throw this; }
        virtual List* tail () { throw this; }
        virtual bool isNil () { return true; }
};
Real example: Linked lists

A concrete class for a cons cell:

class ConsList : public List {
    public:
        ConsList (int hd, List* tl) {
            this->hd = hd;
            this->tl = tl;
        }
        virtual int head () { return hd; }
        virtual List* tail () { return tl; }
        virtual bool isNil () { return false; }
    private:
        int hd;
        List* tl;
};
Real example: Linked lists

class ListStack : public Stack {
  public:
    ListStack () {
      this->contents = List::nil ();
    }
    virtual void push (int x) {
      contents = contents->cons (x);
    }
    virtual int pop () {
      int result = contents->head ();
      contents = contents->tail ();
      return result;
    }
    virtual bool isEmpty () {
      return contents->isNil ();
    }
    virtual bool isFull () {
      return false;
    }
    virtual Stack* copy () {
      ListStack* result = new ListStack ();
      result->contents = contents;
      return result;
    }
  private:
    List* contents;
};

ListStack.cc is on-line.
**OO programming**

OO programming relies on:

- Classes and inheritance
- Heap-allocated objects
- Dynamic method dispatch

It’s not just object-based programming!
Java

Java: an example OO language

Java is an ‘objects-first’ language.

Syntax is C++-like, but cleaner.

Includes features from functional languages (e.g. garbage collection, inner classes).

‘Killer app’ is internet programming.
Pragmatics

Download JDK (Java Development Kit) from course home page.

Install it.

Edit a new file HelloWorld.java:

```java
public class HelloWorld {
    public static void main (String[] args) {
        System.out.println("Hello world");
    }
}
```

To compile, run `javac HelloWorld.java` in a shell.

To run, run `java HelloWorld` in a shell.
Gotchas

DOS shell weirdness: if you get ‘this program cannot run in DOS mode’ then go to the MS/DOS pull-down menu (click on the left-hand MS/DOS icon in the title bar), select Properties/Advanced, and make sure the first checkbox is unchecked.

If you get a message ‘cannot find class file’ then try playing with the CLASSPATH environment variable. It should be:

    set CLASSPATH .;c:\jdk\lib\classes.zip

(assuming you unpacked JDK into c:\jdk\lib)
Structure of a Java program

public class HelloWorld {
    public static void main (String[] args) {
        System.out.println ("Hello world");
    }
}

HelloWorld is the name of the class.

main is the 'bootstrapping' method.

args is the command-line arguments to the program.

System.out.println prints to standard output.
Fields

static fields are one-per-class rather than one-per-object.

final fields cannot be changed (a la functional programming).

public fields are accessible by anyone.

private fields are accessible only by this class.

For example:

```java
public abstract class List {
    public static final List nil = new EmptyList();
    ...
}
public class NilList extends List {
    // no fields!
    ...
}
public class ConsList extends List {
    final private int hd;
    final private List tl;
    public ConsList (int hd, List tl) {
        this.hd = hd; this.tl = tl;
    }
    ...
}
```
Methods

static methods are associated with the class, not the objects of that class.

final methods cannot be overridden.

public methods are accessable by anyone.

private methods are accessable only by this class.

abstract methods do not have a body (only allowed in abstract classes).

For example:

public abstract class List {
    ...
    public List cons (int x) {
        return new ConsList (x, this);
    }
    public abstract int head ();
    public abstract List tail ();
    public abstract List append (List l);
    ...
}
Example cont.

class NilList extends List {
    ...
    public int head () { throw new NoSuchElementException (); }
    public List tail () { throw new NoSuchElementException (); }
    public List append (List l) { return l; }
    ...
}

class ConsList extends List {
    ...
    public int head () { return hd; }
    public List tail () { return tl; }
    public List append (List l) { return tl.append (l).cons (hd); }
    ...
}

Compare with SML:

datatype IntList = Nil | Cons of int * IntList;

List.java, ListStack.java and ListStackDemo.java are on-line.
Another example

How would we complete this?

abstract class BTree {
    public static final BTree leaf = ...;
    public BTree node (BTree left, int root, BTree right) { ... }
    public abstract int root ();
    public abstract BTree leftChild ();
    public abstract BTree rightChild ();
    public abstract int size ();
    public abstract List flatten ();
}

Compare with SML:

datatype BTree = Leaf | Node of BTree * int * BTree;
Java language

Java is call-by-value, statically scoped.

Java supports dynamic method dispatch.

All objects are heap-allocated.

No pointer arithmetic.

No free: uses a garbage collector instead.
Next week

More Java!