CSCI 112
Monday, July 12, 2010

How to Read from Files (Review)

Example Code (Incomplete code, won't compile)

```
import java.io*;
import java.util.*;

Scanner scan = null;
try{
        scan = new Scanner( new File( filename ) );
}
catch( Exception e ){
        System.out.println( "File not found: " + filename );
        System.exit(1);
}

int[][] a = new int[9][9];

for( int i = 0; i < 9; i++ ){
        for( int j = 0; j < 9; j++ ){
                a[i][j] = scan.nextInt();
        }
}
```

Polymorphism

Three Properties of Polymorphism

```
A a = new B(); //B extends A
```

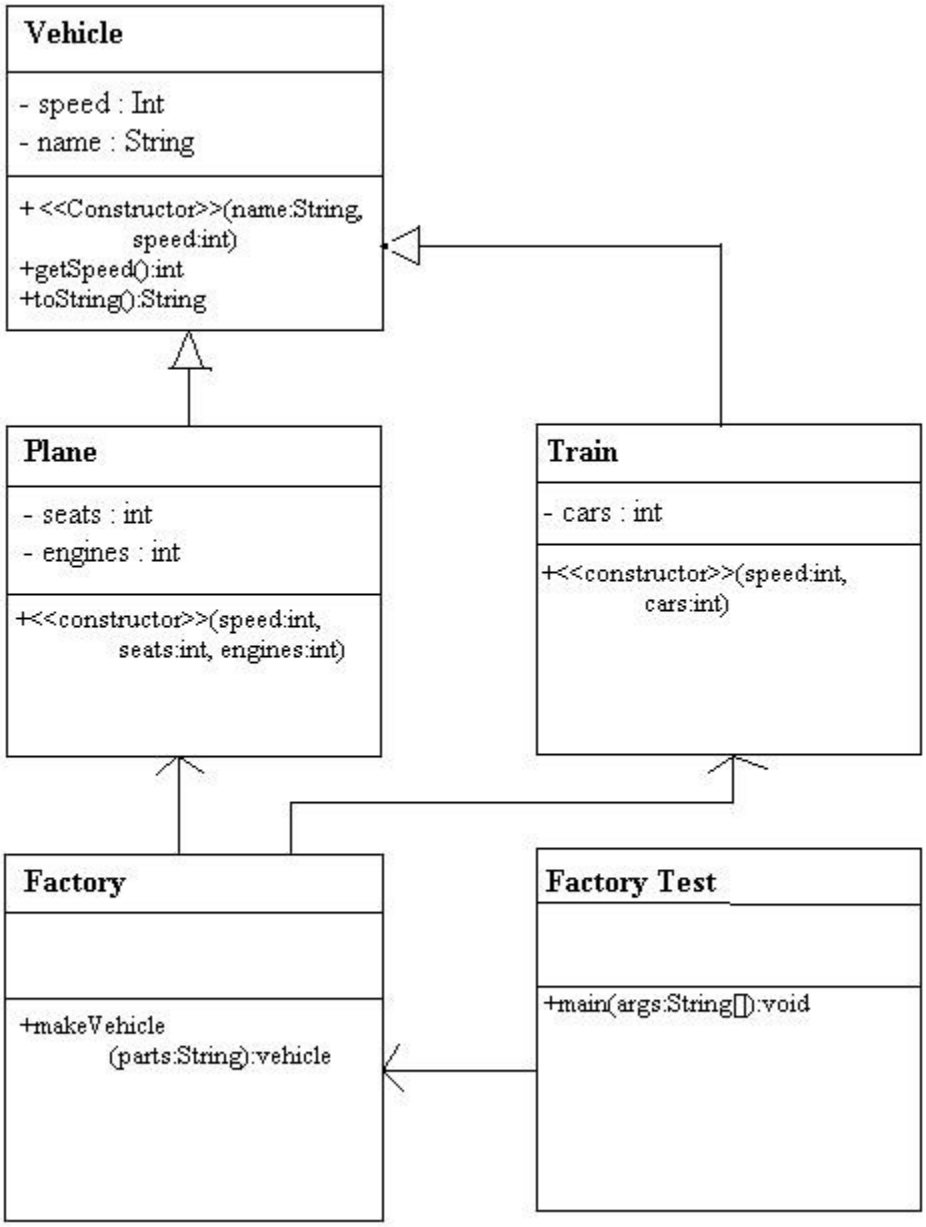
1. "a" may execute all methods in A.
2. "a" will execute overridden methods in B.
3. Methods unique to B are not accessible.

Late Binding – An object is associated with a reference at execution time, not compile time.

The Factory Pattern

- A design pattern developed in 1995 by a group known as the "Gang of Four."
  - The wrote a book called *Design Patterns*.

100712 Table1.jpg – UML Diagram for Example Code



Example Code (Factory Pattern)

parts.txt

```
4
Plane 300 50 2
Train 50 20
Plane 100 2 1
Train 30 10
```

Factory.java

```
import java.util.*;

public class Factory{
	public static Vehicle makeVehicle( String parts ){
		Scanner type = scan.next();

		if( type.equals( "Train" ) ){
			int speed = scan.nextInt();
			int cars = scan.nextInt();

			return new Train( speed, cars );
		}
		if( type.equals( "Plane" ) ){
			int speed = scan.nextInt();
			int seats = scan.nextInt();
			int engines = scan.nextInt();

			return new Plane( speed, seats, engines );
		}
		System.out.println( "Unknown: " + type );
		System.exit(1);
		return new Train( 0, 0 ); //never called
	}
}
```

FactoryTest.java

```
import java.util.*;
import java.io.*;

public class FactoryTest{
	public static void main( String[] args ) throws Exception{
		Scanner scan = new Scanner( System.in );
		System.out.println( "Enter a filename: " );
		String filename = scan.next();

		Scanner fh = new Scanner( new File( filename ) );
		int vehicle_count = fh.nextInt();
		fh.nextLine();

		Vehicle[] vehicles = new Vehicle[vehicle_count];

		for( int i = 0; i < vehicle_count; i++ ){
			String parts = fh.nextLn();

			vehicles[i] = Factory.makeVehicle(parts);
		}
```

```
		for( int i = 0; i < vehicle_count; i++ ){
			System.out.print( vehicles[i] + " " + vehicles[i].getSpeed() );
		}
	}
}
```

Vehicle.java

```
//make code from UML diagram if needed
```

Train.java

```
//make code from UML diagram if needed
```

Plane.java

```
//make code from UML diagram if needed
```

Interfaces

Interface – a collection of constants and abstract methods.

Example Code (Simple Cryptography Program)
Encryptable.java

```
public interface Encryptable{
	public void encrypt();
	public void decrypt();
}
```

Rot13.java

```
public class Rot13 implements Encryptable{
	private String message;
	private boolean encrypted;

	public Rot13( String message){
		this.message = message;

		encrypted = false;
	}
	public String toString(){
		return message;
	}
	private void rotate(){
		String masked = "";

		for( int i = 0; i < message.length(); i++ ){
```

```
                        char ch = message.charAt(i);

                        if( 'a' <= ch && ch <= 'z' ){
                                ch = (char)( 'a' + (( ch - 'a' ) + 13 )%26 );
                        }
                        if( 'A' <= ch && ch <= 'Z' ){
                                ch = (char)( 'A' + (( ch - 'A' ) + 13 )%26 );
                        }
                        masked += ch;
                }
                message = masked;
        }
        public void encrypt(){
                if( encrypted == false ){
                        rotate();
                        encrypted = true;
                }
        }
        public void decrypt(){
                if( encrypted == true ){
                        rotate();
                        encrypted = false;
                }
        }
}
```

RotTest.java

```
public class RotTest{
        public static void main( String[] args ){
                String message = "Attack at Dawn";
                Rot13 r = new Rot13( message );

                System.out.println( r );

                r.encrypt();
                System.out.println( r );
        }
}
```

Interfaces were designed to allow for multiple interfaces to be implemented within one class, because it wouldn't be the same as inheritance and two different interfaces shouldn't contradict each other. One way they can conflict though is if the interfaces contain constants.

Example Code

InterfaceTest.java

```
interface Ia{ final public int X = 2; }
interface Ib{ final public int X = 3; }

class A implements Ia, Ib{
	static void getX(){
		System.out.println(X);
	}
}
class InterfaceTest{
	public static void main( String[] args ){
		A.getX();
	}
}
```

This will not compile, will error: “Cannot compile program, X is ambiguous.” If you comment out one of the interfaces though, it will run just fine.