CS 241 — Introduction to Problem Solving and Programming

Object-Oriented Programming

Subtype Polymorphism

Mar 23, 2005

Interfaces

What happened??

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

Interfaces

meas has type Measurements, but we are assigning it a Measurements1 or Measurements2.

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

Interfaces

At this point, meas could be a Measurements1 or Measurements2.

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

Interfaces

Should the language allow this?

Is this intuitive?

`Measurements1` and `Measurements2` are each a particular variety of `Measurements`, so it makes sense that one could use them in place of a `Measurements`.

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

Interfaces

Should the language allow this?

Can this go wrong?

`Measurements1` and `Measurements2` have the same public members, so any member of `Measurements` referred to has an implementation in the classes.

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

Types

A type is a category of data recognized for two reasons:

1. How the data is stored in the computers memory
2. How the data is used

```
a = x + y;
```

What difference does it make if `x` and `y` are `Strings` or `ints`?

Types and Subtypes

We've seen that classes define types. Interfaces also define types, but only the second category.

A class that implements an interface fills in the details of the first (and possibly adds to the second).

A class that implements an interface defines a subtype of the interface's type.

Types as sets

$$5 \in \mathbb{Z} \subseteq \mathbb{R}$$

$$\mathsf{a} \in \{\mathsf{a}, \mathsf{b}\} \subseteq \{\mathsf{a}, \mathsf{b}, \mathsf{c}\}$$

Think:

$$\texttt{new Measurements1()} \in \texttt{Measurements1} \subseteq \texttt{Measurements}$$

and therefore

$$\texttt{new Measurements1()} \in \texttt{Measurements}$$

Everyday examples

An Escape is a subtype of Ford, and Ford is a subtype of automobile. My Escape is an Escape, it is a Ford, and it is an automobile.

Suppose might be a sophomore computer science student. He is a sophomore, he is a computer science major, and he is a student. Sophomore and computer science major are each subtypes of student.

This is called an is-a relationship.

Types and subtypes

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

What is the type of `Meas`? Depends on what you mean by "what type". . .

Static and dynamic type

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

This will work if `B` is a subtype of `A`.

The static (or declared or compile-time) type of a variable is the type given at its declaration.

The dynamic (or run-time or concrete) type of a variable (or any other expression) is the class of the object to which it refers at a given moment in the program.

$$\text{Dynamic type} \subseteq \text{Static type}$$

Interfaces

```
Measurements meas;
if (DocsIO.readint("Use version 1 or 2?") == 1)
     meas = new Measurements1();
else
     meas = new Measurements2();
meas.add(DocsIO.readdouble("Next reading-->"));
```

The problem is, what add method will be called?

Polymorphism

```
x.m(a, b)
```

When a method invocation is executed, the method in the dynamic type class of the receiver is the one called.

(Only the static types of the parameters are considered.)

The ability for a variable or expression to be treated according to its (possibly varied) dynamic type is called (subtype) polymorphism

polus— many
morphe— shape

Polynomial example

Specification:

Write a class that models a polynomial. The class should support

- Printing the polynomial as a string
- Evaluate the polynomial (as a function) for a value of x
- Compute the derivative (another polynomial)
- Compute the definite integral for given lower and upper bounds.

Function example

Polynomials are not the only structures that have these concepts or functionality.

These operations exist for all (differentiable) functions, of which *polynomial* is a subtype.

New task: write `Rational` and `Step` function classes; let them implement the same interface as `Polynomial`

Function example

```
public interface Function
    public String asString();
    public double evaluate(double x);
    public Function derivative();
    public double integrate(double lower, double upper);
```

Function example

```
public class Polynomial implements Function {
    ...
    public Function derivative() {
        // The array to hold the coefficients for the new polynomial
        double[] newCoefficients = new double[coefficients.length - 1];
        for (int i = 1; i < coefficients.length; i++)
            newCoefficients[i - 1] = coefficients[i] * i;
        return new Polynomial(newCoefficients);
    }
...
```

`derivative()` can return a `Polynomial` even though its return type is `Function` because a `Polynomial` is a `Function`.

Function example

```
public class Polynomial implements Function {
    public Polynomial product(Polynomial other) {
        ...
    }
    public Polynomial sum(Polynomial other) {
        ...
    }
    public Polynomial difference(Polynomial other) {
        ...
    }
}
```

Assume these are written. . . A class can always implement more than its declared interface.

Function example

```
public class Rational implements Function {

    private Polynomial numerator;
    private Polynomial denominator;

    public Rational(Polynomial numerator, Polynomial denominator) {
        this.numerator = numerator;
        this.denominator = denominator;
    }

    public Rational() {
        System.out.println("Numerator:");
        numerator = new Polynomial();
        System.out.println("Denominator:");
        denominator = new Polynomial();
    }
```

Function example

```
public String asString() {
    String numString = numerator.asString();
    String denomString = denominator.asString();
    String toReturn = numString + "/n";
    for (int i = 0; i < numString.length() || i < denomString.length(); i++)
        toReturn += "-";
    toReturn += "/n" + denomString;
    return toReturn;
}

public double evaluate(double x) {
    return numerator.evaluate(x) / denominator.evaluate(x);
}
```

Function example

```
    public Function derivative() {
        Polynomial derivNumerator =
            denominator.product((Polynomial) numerator.derivative())
            .difference(numerator.product((Polynomial) denominator.derivative()));
        Polynomial derivDenominator =
            denominator.product(denominator);
        return new Rational(derivNumerator, derivDenominator);
    }

    public double integrate(double lower, double upper) {
         ...
    }
}
```

Function example

```
public class Step implements Function {

    private double stepPoint;
    private double stepLevel;

    public Step(double stepPoint, double stepLevel) {
        this.stepPoint = stepPoint;
        this.stepLevel = stepLevel;
    }

    public Step() {
        stepPoint = DocsIO.readdouble("Step point--> ");
        stepLevel = DocsIO.readdouble("Step level--> ");
    }
```

Function example

```
public String asString() {
    return 0 + " if x < " + stepPoint + ", " + stepLevel + " otherwise";
}

public double evaluate(double x) {
    if (x < stepPoint)
        return 0;
    else
        return stepLevel;
}
```

Function example

```
    public Function derivative() {
        double[] zero = { 0.0 };
        return new Polynomial(zero);
    }

    public double integrate(double lower, double upper) {
        if (lower == upper) return 0;
        else if (lower > upper) return - integrate(upper, lower);
        else if (upper < stepPoint) return 0;
        else if (lower > stepPoint) return stepLevel * (upper - lower);
        else return stepLevel * (upper - stepPoint);
    }
}
```

Function example

```
public class FunctionDriver {
    public static void main(String[] args) {
        // The function on which we run our tests
        Function test;
        int choice = DocsIO.readint("1=Polynomial, 2=Rational, 3=Step");
        if (choice == 1)
            test = new Polynomial();
        else if (choice == 2)
            test = new Rational();
        else
            test = new Step();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());
```

Function example

```
    public Function derivative() {
        double[] zero = { 0.0 };
        return new Polynomial(zero);

        // Value on which we'll evaulate the function
        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        // Bounds for the definite integral
        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Think about. . .

Library example

Books, magazines, journals, recordings. . .

Faculty, staff, students. . .

Summary

- Type
- Subtype
- Static type
- Dynamic type
- Polymorphism