CS 241 — Introduction to Problem Solving and Programming

Object-Oriented Programming

Second look at classes

Mar 14, 2005

Types, expressions, statements

A type is a categorization of data (`int`, `String`, `char[]` . . .).

A variable is a container for storing data; since it must be appropriate for the data being stored, it has a type.

An expression is a construct in the language which has a value; it evaluates to something; therefore, it also has a type.

A statement is a construct in the language which is executed and may have some side effects (printing to the screen, setting a variable) but does not produce a value.

You can turn an expression into a statement by appending a semi-colon.

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Types, expressions, statements

Is it an expression (and what type?) or a statement?

```
time.am = (DocsIO.readint("0 = AM, 1 = PM") == 0);
```

Methods

A method is

- A broken-off piece of the algorithm
- A packaged/encapsulated piece of functionality
- A module that can be reused like interchangeable parts
- A machine with slots going in and a slot coming out

Methods

Some examples. . .

- `static double convert(double temp)`
- `static int gcd(int a, int b)`
- `static int rollDice(int numDice, int numRolls, int monitorNumber)`

Methods

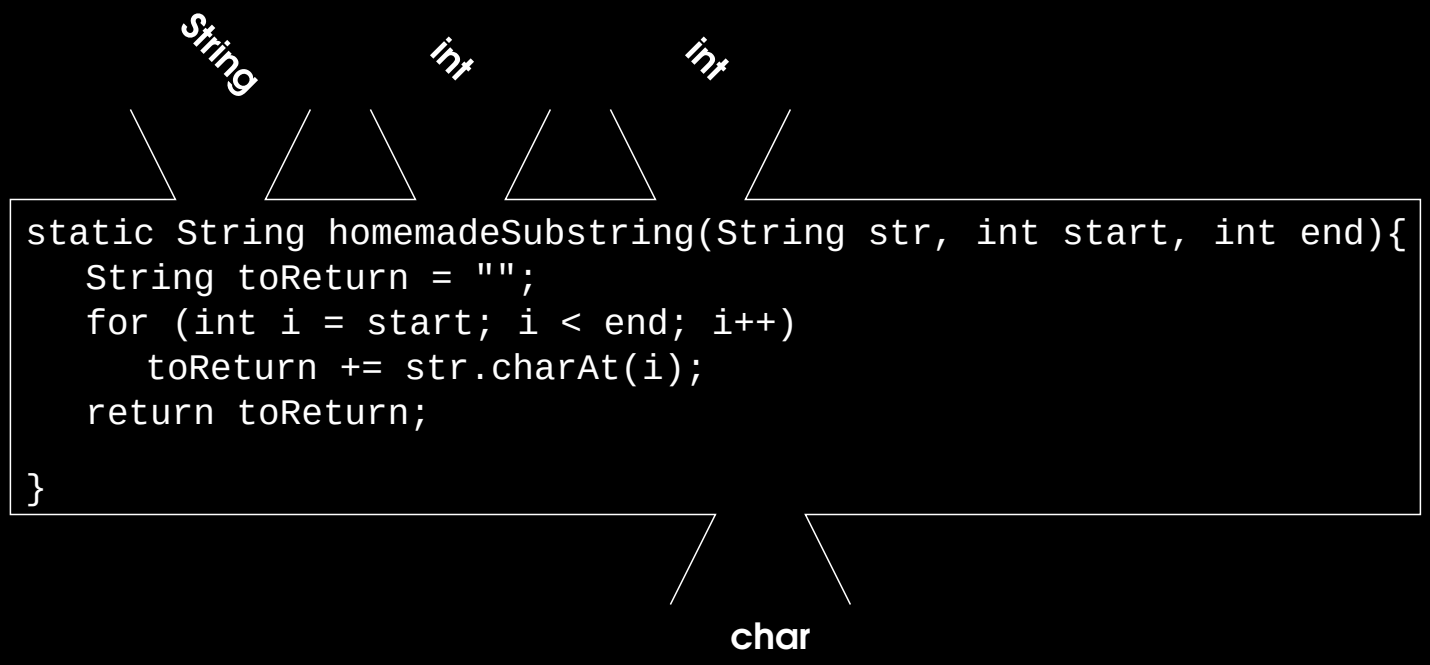
Methods are like variables that calculate a value rather than retrieve one from storage:

```
int gcd;

...

static int gcd(int a, int b)
```

A method call, like the use of a variable, is an expression and so it has a type.

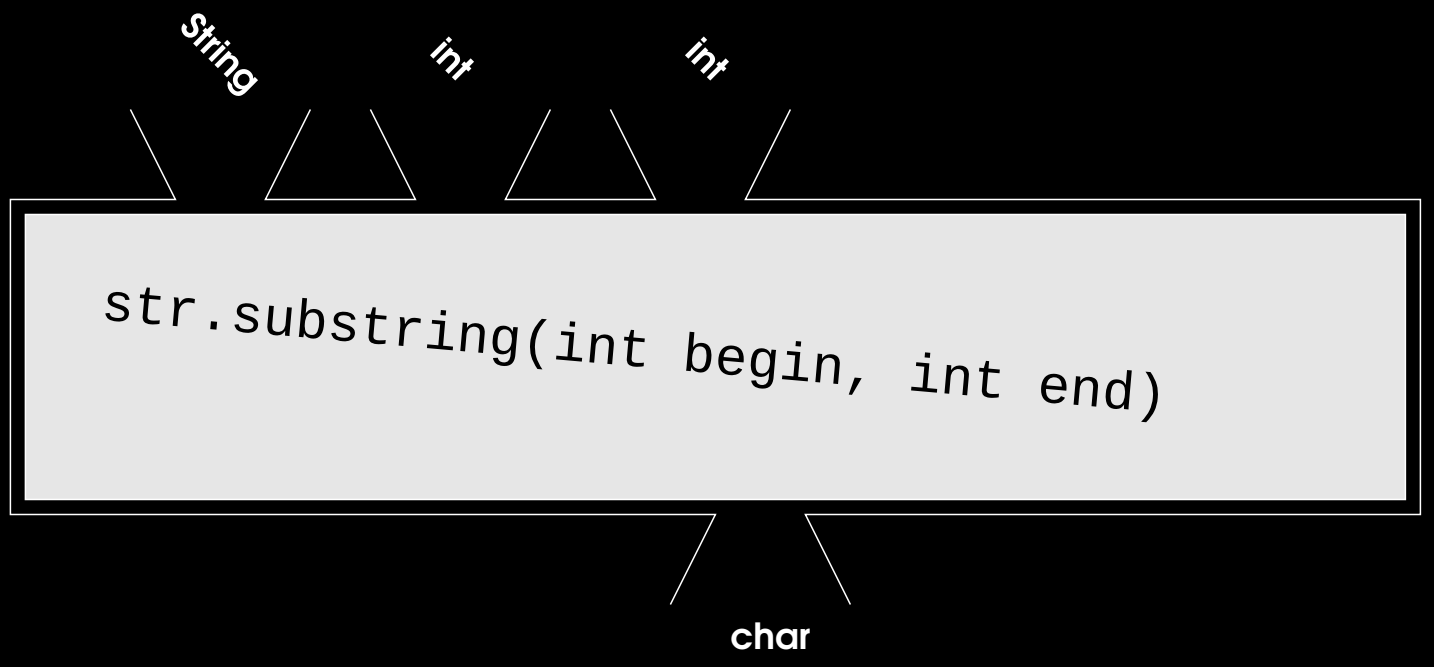
Methods

We have thought of methods as machines. You can feed things into the machine (the parameters), and the machines will produce something (what is returned).



Methods

But sometimes we've used methods without seeing how they work—like machines with a cover on them (ie, black boxes).



Reference/composite types

Recently we've seen new types:

```
int[] intArray = new array[50];

class Point {
   double xCoord, yCoord;
   String label;
}

Point point = new Point();
point.xCoord = 0;
point.yCoord = 0;
point.label = "origin";
```

These are composite types because they are composed for several parts. They are reference types because their variables and expressions refer to objects in memory.

Classes

Some examples. . .

```
class Time {
   int year;
   String month;
   int date;
   boolean am;
   int hour;
   int minute;
}
```

```
class Book {
    String title;
    String author;
    int pages;
    int callNumber;
    Patron user;
    int daysTillDue;
}
```

Classes

Classes package or encapsulate data.

. . . but, in order to use them, we need direct access to the instance variables—no black-box use.

Moreover, some things we want to model seem a little like a set of data, a little like a set of functionality (do we use a method or a class?).

Example: Watch class

Classes

Goal: Package data and functionality together.

Now think of an object as a set of data and operations on that data.

Classes

```
class A {
    int data;
    void incrementData() {
        data++;
    }
    int getDouble() {
        return data * 2;
    }
}
```

```
A a = new A();
a.data = 5;
a.incrementData();
a.incrementData();
System.out.println(a.getDouble());
```

Classes

```
class A {
    int data;
    void incrementData() {
        data++;
    }
    int getDouble() {
        return data * 2;
    }
}
```

A class has data components and method components.

Classes

Both types of components are used through dot notation.

```
A a = new A();
a.data = 5;
a.incrementData();
a.incrementData();
System.out.println(a.getDouble());
```

Classes

```
class A {
    int data;
    void incrementData() {
        data++;
    }
    int getDouble() {
        return data * 2;
    }
}
```

The data components can be used in the body of the method components. They refer to the data for that instance.

Thus they are called instance variables.

Distinguish them from local variables, local to a specific block.

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.

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Create a new polynomial.

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Display it.

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Find the derivative.

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Evaluate it at a certain point.

Polynomial example

```
public class PolynomialDriver {
    public static void main(String[] args) {
        Polynomial test = new Polynomial();
        System.out.println(test.asString());
        System.out.println(test.derivative().asString());

        double value = DocsIO.readdouble("Test value: ");
        System.out.println(test.evaluate(value));

        double lowerBound = DocsIO.readdouble("Lower bound: ");
        double upperBound = DocsIO.readdouble("Upper bound: ");
        System.out.println(test.integrate(lowerBound, upperBound));
    }
}
```

Find a definite integral

Polynomial example

```
public class Polynomial {

    /**
     * The coefficients of this polynomial, stored as an array of doubles.
     * The index into the array is the same as the degree of the
     * term for which that position is the coefficient. For example,
     * the constant term is coefficients[0], and the coefficient for
     * the 3rd-degree term is coefficients[3]. Hence the degree of
     * the polynomial is inferred from the length of the array.
     */
    double[] coefficients;
```

Instance variables

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

A constructor is a special method for initializing the instance of a class. Its name is the same as the class; it has no return type (not even `void`).

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

It is called when you instantiate the class—that's why we have those parentheses after the class name when we instantiate.

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

Constructors can have parameters.

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

Constructors (or any other methods) can have parameters with the same identifier as an instance variable. In this case the parameter shadows the instance variable.

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

All classes have a default instance variable called `this`. It is a reference to itself.

this

A class can use `this` to pass a reference to itself to a method or return it from a method.

```
void printGraph() {

    Grapher grapher = new Grapher();
    ...
    grapher.drawGraph(this);
}
```

Polynomial example

```
/**
 * Constructor based on a given set of coefficients.
 * @param coefficients The array holding the values to become this
 * polynomial's coefficients
 */
Polynomial(double[] coefficients) {
    this.coefficients = coefficients;
}
```

`this` can also be used to refer to shadowed instance variables.

Polynomial example

```
/**
 * Constructor without a parameter on which to base the polynomial.
 * Get data from the user instead.
 */
Polynomial() {
    int degree = DocsIO.readint("What is the degree of this polynomial? ");
    coefficients = new double[degree + 1];
    for (int i = 0; i < coefficients.length; i++)
        coefficients[i] =
            DocsIO.readdouble("What is the coefficient of the term of degree "
                              + i + "?");
}
```

Constructors can be overloaded.

Constructors

How did we instantiate classes before we knew how to write constructors?

If you do not write a constructor for a class, Java provides a default constructor which has no parameters and initializes all references to `null`, all `int`s to 0, all `boolean`s to `false`. . .

If you write any constructor, the default constructor will not be produced.

Polynomial example

```
/**
 * Make a string representation of the polynomial consistent with how we
 * normally write polynomials.
 * @return This polynomial as a string.
 */
String asString() {
    String toReturn = "";
    for (int i = coefficients.length - 1; i > 0; i--)
        toReturn += coefficients[i] + " x-" + i + "  +  ";
    toReturn += coefficients[0];
    return toReturn;
}
```

Polynomial example

```
/**
 * Make a string representation of the polynomial consistent with how we
 * normally write polynomials.
 * @return This polynomial as a string.
 */
String asString() {
    String toReturn = "";
    for (int i = coefficients.length - 1; i > 0; i--)
        toReturn += coefficients[i] + " x-" + i + "  +  ";
    toReturn += coefficients[0];
    return toReturn;
}
```

Local variable.

Polynomial example

```
/**
 * Make a string representation of the polynomial consistent with how we
 * normally write polynomials.
 * @return This polynomial as a string.
 */
String asString() {
    String toReturn = "";
    for (int i = coefficients.length - 1; i > 0; i--)
        toReturn += coefficients[i] + " x-" + i + "  +  ";
    toReturn += coefficients[0];
    return toReturn;
}
```

Instance variable

Polynomial example

```
/**
 * Evaluate the function of this polynomial for a give value of x.
 * @param x The value at which to evaluate the polynomial.
 * @return The value of this polynomial at x
 */
double evaluate(double x) {
    double toReturn = 0;           // To hold our result as we accumulate
    double currentPower = 1;       // The power of x at the current degree
    for (int i = 0; i < coefficients.length; i++) {
        toReturn += coefficients[i] * currentPower;
        currentPower *= x;
    }
    return toReturn;
}
```

Polynomial example

```
/**
 * Find the derivative of this polynomial.
 * @return A new polynomial, the derivative of this one
 */
Polynomial 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);
}
```

Polynomial example

```
/**
 * Find the derivative of this polynomial.
 * @return A new polynomial, the derivative of this one
 */
Polynomial 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);
}
```

Local variable

Polynomial example

```
/**
 * Find the derivative of this polynomial.
 * @return A new polynomial, the derivative of this one
 */
Polynomial 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);
}
```

Instance variable

Polynomial example

```
/**
 * Find the derivative of this polynomial.
 * @return A new polynomial, the derivative of this one
 */
Polynomial 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);
}
```

Constructor call

Polynomial example

```
    /**
     * Calculate the definite integral of this polynomial, given upper and lower
     * @param lower The x value at which the region begins; the lower bound
     * @param upper The x value at which the region ends; the upper bound
     * @return The value of the definite integral
     */
    double integrate(double lower, double upper) {
        // The array to hold the coefficients for the anti-derivative
        double[] newCoefficients = new double[coefficients.length + 1];
        for (int i = 1; i < newCoefficients.length; i++)
            newCoefficients[i] = (1 / (double) i) * coefficients[i - 1];
        // The anti-derivative of this polynomial
        Polynomial integral = new Polynomial(newCoefficients);
        return integral.evaluate(upper) - integral.evaluate(lower);
    }
}
```

Summary

Be familiar with the following concepts.

- Type
- Variable
- Expression
- Statement
- Method
- Composite or reference types
- Members
- Data members
- Method members
- Instance variables
- Local variables
- Constructor
- Default constructor
- Shadowing / shadowed variables
- `this`