Objects

CSE 142, Summer 2002

Computer Programming 1

http://www.cs.washington.edu/education/courses/142/02su/

Readings and References

- Reading
  - Chapter 2, *An Introduction to Programming and Object Oriented Design using Java*, by Niño and Hosch
  - Chapter 4, *Introduction to Programming in Java*, Dugan
- Other References

Objects

- Objects are the fundamental unit of our programs
- An object has
  - State
  - Behavior
- The state of an object is described by one or more values
- For example, an object describing a student might contain values for the following properties
  - name, home address, credit hours, UWNetID, course schedule
- The behavior of an object is described by the *methods* that it implements

Values

- What types of values are there?
  - Lots and lots, gazillions, tons, a hunka values, …
- There are many different values that we deal with every day
  - some are simple:
    credit hours, price, UWNetID
  - some are more complex:
    university course, building, tax return
  - some are very complex:
    graphics window, web site, file system, classroom assignments

Integers

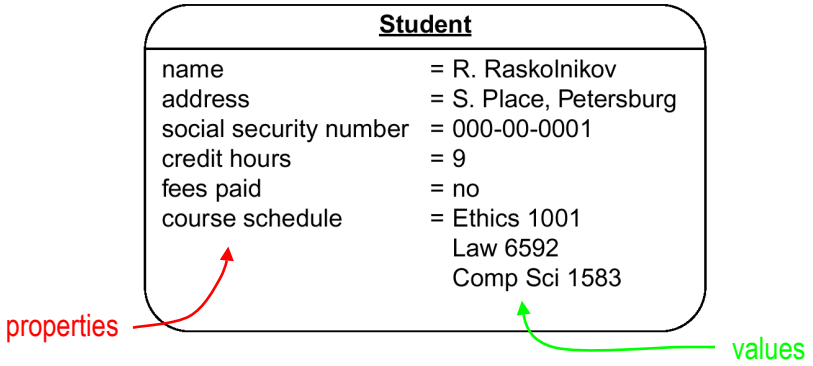
- The simplest set of values are the integers
- The integers are whole numbers
  - there is no fractional part in an integer value
- For example: …, -2, -1, 0, 1, 2, … ,100, … 102394, …
  - note that it takes more room to store big numbers
- Java provides several types of integers, so that we can use only the space we need
  - Don't be too worried about saving space, memory is cheap
  - We'll use integers of type **`int`**

    smallest int = -2,147,483,648

    largest int = 2,147, 483,647

Storing values

- The state of the object is described by the set of values that are assigned to its properties



Declaring properties (or variables)

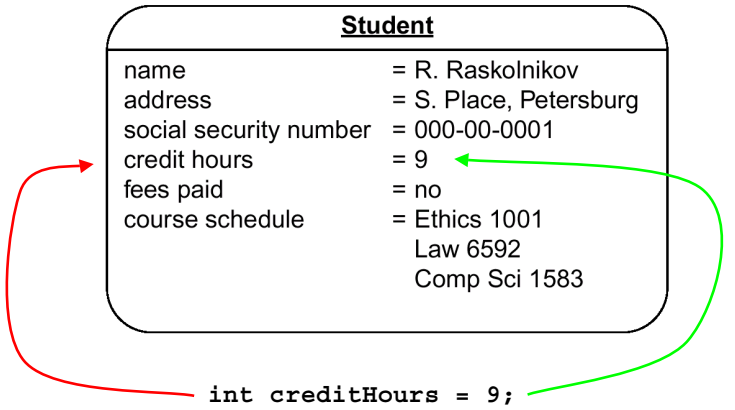
- In order to define a property and give it a value, we have to *declare* it
- This gives the property a *name* and a *type* so that we can assign it a *value*
- The pattern of a declaration is

  <the type of thing> <the name> = <the value>;
- An example of this pattern

```
int width = 10;
int height = 5;
```

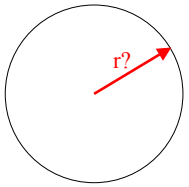
- The object has declared the properties width and height
- The values of those properties are 10 units and 5 units

Declaring a variable



But not all numbers are integers!

- What is the radius of a circle?
  2.75 cm



- How many miles to the gallon does your Honda Accord get?
  33.5 miles per gallon
- What is the area of a circle?

$$Area = \pi \cdot Radius^2$$

- What is the value of π?
  π = 3.1415926535...

Floating Point Numbers

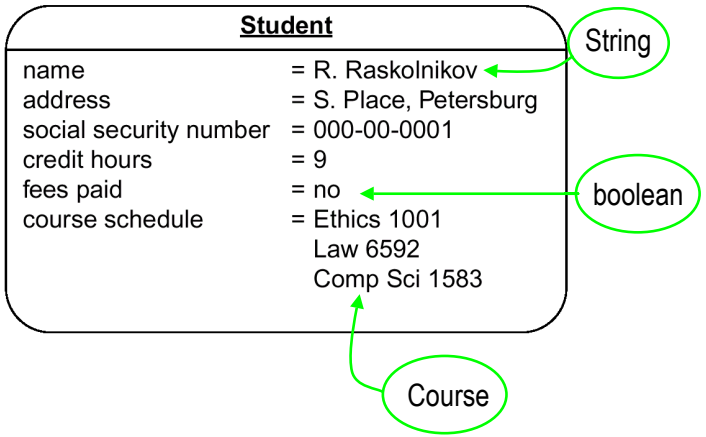
- Java uses "floating point numbers" to store values that cannot be represented as `int`s
  - numbers with a fractional part
  - very very very large numbers
  - 2.75, 33.5, 3.14159, $2.3 \cdot 10^{120}$
- Many numbers like $\frac{1}{3}$ and π cannot be represented exactly in the number system of the computer, and so they are approximated $\frac{1}{3} \approx 0.3333$ $\pi \approx 3.14159$

Declaring Floating Point Properties

- Java provides two types of floating point numbers
  - float
  - double
- We will use type double
  - it uses more memory space, but what the heck, memory is cheap
- For example

```
double radius = 2.75;
double PI = 3.14159;
double area = PI*radius*radius;
```


But not all values are numbers!



`boolean` values are `true` or `false`

- A boolean variable is appropriate for properties that you know can be in only one of two logical states
  - admitted or not admitted to school
  - registered or not registered for the class
  - fees paid or not paid
  - window hidden or not hidden
- declaration examples

```
boolean feesPaid = false;
boolean windowHidden = true;
```

- Notice that it's not always obvious that a property can only be true or false under all circumstances

String values

- We often want to store a sequence of characters together
- This is called a "String"
- Declaration pattern example

```
String name = "Doug Johnson";
String city = "Langley";
```

- Strings are full fledged objects
  - They have state

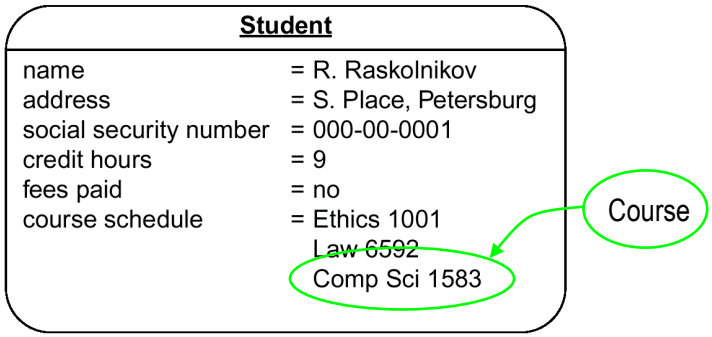
```
length(), charAt(int index), endsWith(String suffix),…
```

  - They have behavior

```
concat(String str), compareTo(String anotherString),…
```


And then the fun begins ...

- What if we want to define and use something that is not one of the 2723 standard Java classes?
- For example, what exactly is a "Course"?



What is a Course?

- We can *model* a course any way that we feel is appropriate for the application we are developing.
- What are the properties?
- What are the behaviors?
- Maybe we don't expect much of the Course object. We could just use a String to store the name of the course

```
String course = "Comp Sci 1583";
```

- Maybe we want to store the department name separate from the course number

```
String dept = "Comp Sci";
int number = 1583;
```

What is a Course?

- What are the fundamental properties of the object that you need to keep track of?
  - Figuring this out is one of the fun parts of software design
  - What are the key properties of the object we are designing?

    Yes: course content, course textbook, instructor, lecture schedule

    Maybe: course location, max student count, student list

    Probably not: room facilities, other classes by this instructor
  - What are key behaviors?

    getTextbook(), getLectureList();

    setInstructor(Instructor abc);

    addStudent(Student s);

    …

A real Course has complex properties

- a room assignment
  - the properties of the room are not really properties of the course itself, so we probably have a different class of objects that are "Room" descriptions
- an instructor
  - the details probably belong in an "Instructor" object
- a set of students
  - the details about each student probably belong in a "Student" object
- and so on

We can define our own classes of objects

- The power of Java and other object oriented languages comes from our ability to define new classes of objects that match the needs of the application we are writing
- We can define and use our own classes to describe a Course
  - then we can declare variables (properties) using those classes
- For example

```
Instructor teach = new Instructor("Doug Johnson");
Room room = new Room("EE1", 105);
ArrayList student = new ArrayList();
```

- We will spend a fair amount of time this quarter talking about how we define and use these new classes of objects

Shape Objects

- Many graphics-oriented programs manipulate *shapes*
- Let's create some shapes and windows:

  new Triangle( )

  new Rectangle(200, 50, 100, 10) *(left x, top y, width, height)*

  new GWindow( )
- We use the following patterns for creating new objects:

  new <type of object>(<optional list of parts or attributes>)
- We usually should give newly created objects a name:

  GWindow w = new GWindow( );

  Rectangle kaneHall =

  new Rectangle(50, 150, 250, 200, Color.red, true); *(x, y, w, h, color, filled?)*

  Oval sun = new Oval(200, 50, 35, 35, Color.yellow, true); *(x, y, w, h, color, filled?)*

Sending Messages

- We get objects to do things, or answer questions, or calculate results for us, by *sending them messages*
  - Also called *invoking a method* or (in other languages) *calling a function*
- We use the following pattern for sending a message:

  <object name> **.** <message name> **(** <optional list of parameters> **)**
- Examples:

  sun . getX ( )

  sun . addTo (w)

  sun . moveBy (30, -20)

Drawing a Scene

- To draw a nice picture, first create a window:

  GWindow w = new GWindow( );
- Then create a shape object, and add it to the window:

  Line horizon = new Line(50, 200, 200, 200, Color.green); *(x1, y1, x2, y2, color)*

  horizon.addTo(w);
- Create and add more shapes:

  Oval sun = new Oval(100, 175, 35, 25, Color.orange, true); *(x, y, w, h, c, f?)*

  sun.addTo(w);

  Rectangle deadTree = new Rectangle(150, 150, 10, 50); *(x, y, w, h)*

  deadTree.addTo(w);

  Rectangle tallBuilding = deadTree;

Appendix

Class SimplePicture

```
import uwcse.graphics.*;
import java.awt.Color;
/**
 * This class implements the code given on the last few slides
 * of lecture C1.
 *
 * @author Doug Johnson
 */
public class SimplePicture {

    /** the on-screen window we are drawing in */
    GWindow frame;

    /** the horizon line */
    Line horizon;
    /** jolly mister sun */
    Oval sun;
    /** a simple stump */
    Rectangle deadTree;
```

Class SimplePicture

```
    /**
     * Construct a simple graphics window with all default characteristics.
     */
    public SimplePicture()
    {
        frame = new GWindow();
        horizon = new Line(50, 200, 200, 200, Color.green);
        horizon.addTo(frame);
        sun = new Oval(100, 175, 35, 25, Color.orange, true);
        sun.addTo(frame);
        deadTree = new Rectangle(150, 150, 10, 50,Color.gray, true);
        deadTree.addTo(frame);
    }
    /**
     * This method moves the sun a little bit.
     */
    public void advanceSun()
    {
        sun.moveBy(2,-5);
    }
}
```