

CSC 221: Computer Programming I

Fall 2005

Objects and classes: a first pass

- software objects, classes, object-oriented design
- BlueJ IDE, compilation & execution, *shapes example*
- method calls, parameters
- data types, object state
- object interaction, *picture example*
- other examples: *Die, SequenceGenerator, Singer*

1

Object-oriented programming

the *object-oriented* approach to programming:

- solve problems by modeling real-world objects
e.g., if designing a banking system, model clients, accounts, deposits, ...
- a program is a collection of interacting objects
- in software, objects are created from classes
the class describes the kind of object (its properties and behaviors)
the objects represent individual instantiations of the class

REAL WORLD CLASS: automobiles

REAL WORLD OBJECTS: my 2003 Buick Rendezvous, the batmobile, ...

- the class encompasses all automobiles
they all have common properties: wheels, engine, brakes, ...
they all have common behaviors: can sit in them, start them, accelerate, steer, ...
- each car object has its own specific characteristics and ways of producing behaviors
my car is white & seats 7; the batmobile is black & seats 2
accelerating with V-6 is different than accelerating with jet engine

2

Shape classes and objects

a slightly more abstract example involves shapes

- class: circles
what properties do all circles share?
what behaviors do all circles exhibit?

- objects:

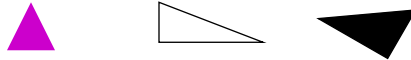


similarly, could define classes and object instances for other shapes

- squares:



- triangles:



3

BlueJ and software shapes

the BlueJ interactive development environment (IDE) is a tool for developing, visualizing, and debugging Java programs

- BlueJ was developed by researchers at Deakin University (Australia), Maersk Institute (Denmark), and University of Kent (UK)
- supported by Sun Microsystems, the developers of Java
- note that BlueJ does NOT include a Java compiler/interpreter
must install Sun's Java SDK (software development kit); BlueJ connects to it
BlueJ includes an editor, debugger, visualizer, documentation viewer, ...

we will start with a visual example in BlueJ: drawing shapes

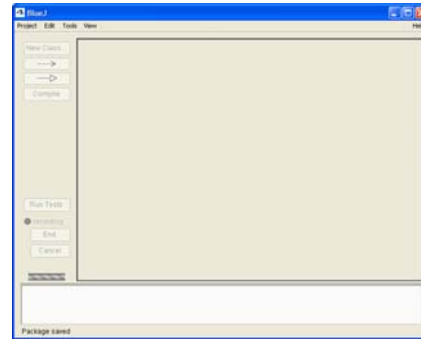
4

Starting up BlueJ

to start up the BlueJ IDE, double-click on the BlueJ desktop icon

this opens the BlueJ main window

- in order to create and execute a program, must first create or load a *project*
- a project groups together all the files needed to produce a working program



to open an existing BlueJ project

- click on the `Project` heading at the top left
- from the resulting pull-down menu, select `Open Project`
- browse to locate and select the project

5

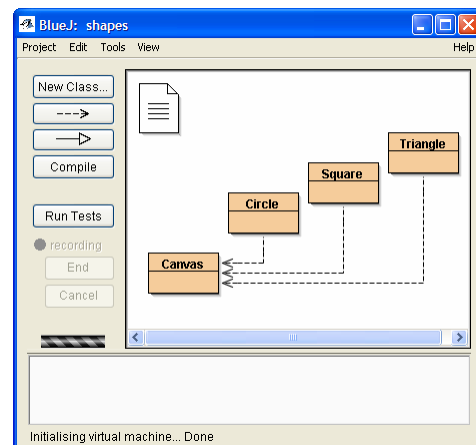
Loading the shapes project

BlueJ comes with a collection of example projects

- click on the `Project` heading
- select `Open Project`
- browse to select `C:\Program Files\BlueJ\examples\shapes`

when a project loads, its classes are shown in a diagram

- here, there are 4 classes
- `Canvas` represents a painting area
- `Circle`, `Square`, and `Triangle` represent shapes
- the arrows show that the shapes depend upon the `Canvas` class



6

Editing and compiling classes

you can view/edit a class definition by double-clicking on its box

- this opens the associated file in the BlueJ editor

before anything can be executed, the classes must be compiled

- recall, the Java compiler translates Java source code into Java byte code
- to compile all classes in a project, click on the Compile button
(note: non-compiled classes are shaded, compiled classes are not)

IMPORTANT: classes don't act, objects do!

- you can't drive the class of all automobiles
- but you can drive a particular instance of an automobile

in order to draw a circle, must create a circle object

- then, can specify properties of that instance (radius, color, position, ...)

7

Example: creating a circle

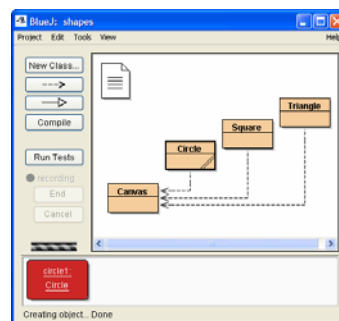
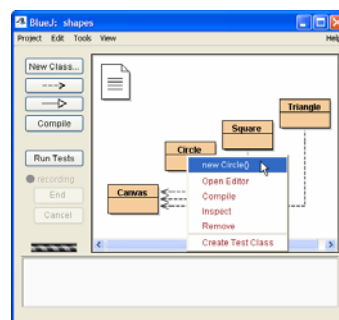
right-click on a class to see all the actions that can be applied

- select `new Circle()` to create a new object
- you will be prompted to specify a name for that object (circle1 by default)

the new Circle object appears as a box at the bottom of the screen

- note: classes and objects look different

EXERCISE: create 2 circles, a square, and a triangle

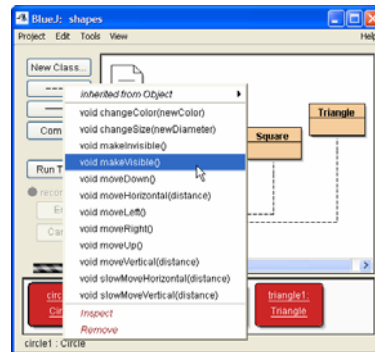


8

Applying object methods

can cause objects to act by right-clicking on the object box, then selecting the action

- the actions that objects can perform are called *methods*

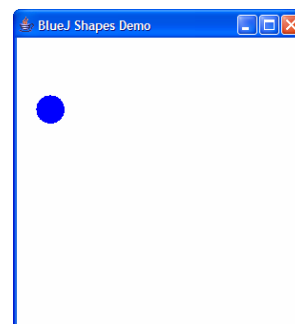


- here, `void makeVisible()` opens a Canvas in which the shape is displayed

EXERCISE: make the other shapes visible

EXERCISE: select other methods to change the color and size of objects

EXERCISE: play



9

Methods and parameters

sometimes an action (i.e., method) requires information to do its job

- the `changeColor` method requires a color ("red", "green", "black", ...)
- the `moveHorizontal` method requires a number (# of pixels to move)
- data values provided to a method are called *parameters*

Java provides for different types of values

- `String` is a sequence of characters, enclosed in double-quotes (e.g., "red")
- `int` is an integer value (e.g., 40)
- `double` is a real value (e.g., 3.14159)
- `char` is a character value (e.g., 'A')
- the parameter to `changeColor` is a `String` representing the new color
- the parameter to `moveHorizontal` is an `int` representing the # of pixels to move

10

Objects and state

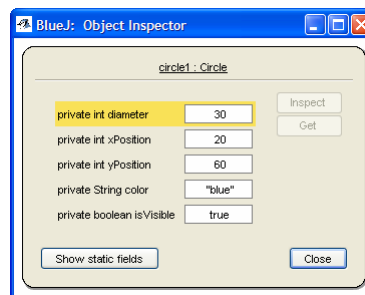
recall that each object has properties and methods associated with it

- when you create a Circle, it has an initial size, color, position, ...
- those values are stored internally as part of the object
- as methods are called, the values may change
- at any given point, the property values of an object define its *state*

BlueJ enables you to inspect the state of an object

- right-click on the object
- select Inspect to see the values of object properties

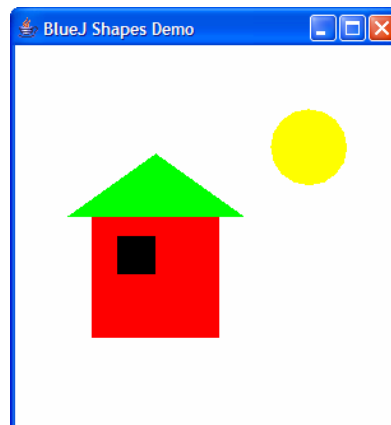
note: objects of the same class have the same properties, but may have different values



11

IN-CLASS EXERCISE

create objects and call the appropriate methods to produce a picture like this



12

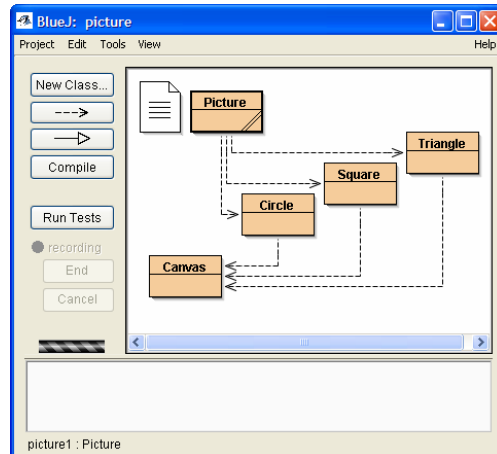
The Picture class

now load the `Picture` project from the examples directory

- the `Picture` class automates the drawing of the house picture
- when the `Draw` method is called on a `Picture` object, the house picture is drawn

EXERCISE: view the source code of `Picture` by double-clicking on its box

EXERCISE: after the line
`sun.makeVisible(); add`
`sun.slowMoveVertical(300);`
then save (Ctrl-S) and Compile



13

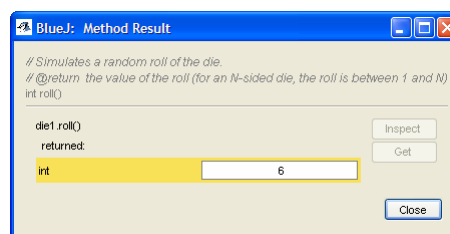
Class examples: Die & SequenceGenerator

can define a `Die` class to model different (numeric) dice

- properties shared by all dice: number of sides, number of times rolled
- behaviors/methods shared by all dice: roll it, get # of sides, get # of rolls

the `roll` method generates a random roll and *returns* it

the return value is displayed by BlueJ in a *Method Result* window



the `SequenceGenerator` class similarly returns a random string of letters

- many interesting problems involve decisions based on random values
- we can use an N-sided `Die` object to select between N alternatives

14

Textual example

can define a `Singer` class to “sing” various children’s songs

- instead of displaying shapes in a Canvas or returning a value, the methods of the `Singer` class display text in a window
- methods include:
 - oldMacDonaldSong()* for “Old MacDonald Had a Farm”
 - bottlesSong()* for “100 Bottles of Dew on the Wall”
 - busSong()* for “The Wheels on the Bus”
- each song utilizes a parameterized method to display a single verse
 - oldMacDonaldVerse(animal, sound)*
 - bottlesSong(numBottles, drink)*
 - busSong(busPart, partAction)*