CSE 1720

Lecture 9

Inheritance, III

Topics

- object serialization
- transformation of drawing space (rotate, shear, …)

Transient vs Persistent Representation of Objects

- we already know and understand that the VM makes use of heap space, which is ***transient*** memory
  - ***transient*** since it lasts only as long as the program executes
- Objects represented in heap space are transient; when the app terminates, the object's lifecycle will come to an end
- what if we want our program to make use of more persistent storage? what are our options?
  - we must read/write to a file

Object Serialization

- as we know, each object has its state, that is the values of each of its non-primitive attributes ("set 1")
  - the value of each of these non-primitive attributes is an object, which itself has a state ("set 2")
  - the values of each of the non-primitive attributes of the aggregated objects may in turn have state ("set 3")
  - and so on… this recursion will ultimately yield attributes that are primitive or string
- this process of recursing into the attributes is called *serializing*
- so to save an object means to serialize it

Example

- suppose an app builds up a set of Shape objects, using the object `Set<Shape>` to represent them.
- How can the object `Set<Shape>` be saved to a file so that it can be read into an app later on?

File I/O

Use services of `File` to encapsulate a file from the file system:

```
String myPathName = File.separator + "Users" +
       File.separator + "mb" +
              File.separator + "myObject.obj";
File myFile = new File(myPathName);
```


File I/O

The class `Scanner` provides services to encapsulate a stream of data and to read the data:

```
Scanner input = new Scanner(myFile);
```

- we already know how to iterate over the tokens/lines in the file, via the `hasNextLine()` and `nextLine()` methods
- the return type of these methods tell us the data types that can be read from the file
- look at API and see that various types can be read: String, int, float, boolean, etc, but no other types such as `Set<Shape>`

File I/O

The class `FileInputStream` provides services to encapsulate a stream of input from a file (deliver its contents as bytes)

```
FileInputStream fis= new FileInputStream(myFile);
FileInputStream fis2 = new FileInputStream(myPathName);
// in the constructor, can use the file object or the string name
of the file
```

The class `ObjectInputStream`

The class `ObjectInputStream` provides services to encapsulate a stream of input from a file (and deliver its contents as objects)

```
ObjectInputStream ois= new ObjectInputStream(fis);
// in the constructor, can use a file input stream
```

The class `ObjectInputStream` provides the following method

Object	**readObject**() Read an object from the ObjectInputStream.

The class `ObjectInputStream`

So if we know the file `myObj.obj` contains an object of type `Set<Shape>`, how do we read this object so that we can use it?

Approach #1

```
Object obj1 = ois.readObject();
Set<Shape> theSet = (Set<Shape>) obj1;
```

Approach #2

```
Set<Shape> theSet = (Set<Shape>) ois.readObject();;
```


Casting is potentially problematic

The manual cast is a weak point in the process.

Manual casts are an opportunity for a run time error.

It is prudent to set up safeguards before performing the cast.

See ex `L09App02`

Iterating over a collection read from a file

See ex `L09App03`

Iterating over a collection

E.g., see method `getShapeCollection(int, int)` from `L09Utility`

See ex `L09App04`

The `Graphics2D` API

- a `Graphics2D` object encapsulates the drawing region (in *device space*) and a set of supported drawing operations on that drawing region (in *user space*)
- All methods can be divided into two groups:
  - Methods to draw a shape
  - Methods that affect rendering

Seven primary attributes of rendering (for Java 2D Graphics)

- Fill (Paint)
- Stroke
- Font
- Transformation
- Clipping space
- Rendering hints
- Compositing rule

Transformation

http://docs.oracle.com/javase/tutorial/2d/advanced/transforming.html

```
public void rotate(double theta);

public void rotate(double theta,
      double aroundPointX, double aroundPointY);

public void scale(double scaleX, double scaleY);

public void shear(double shearX, double shearY);

public void translate(double translateX,
      double translateY);
```

There is also the method
`public void transform(AffineTransform transform)`
We will not discuss `AffineTransform` objects at this point

Transformation

- Important point: it is **drawing surface** that is transformed, not the graphic primitives (the shapes) themselves.
- When the primitives are rendered on the transformed surface, their appearances are altered

Rotation

See ex `L09App05`

Scale

See ex `L09App06`

Shear

See ex `L09App07`

Translate

See ex `L09App08`