CSE 1720

Lecture 5

Aggregation, Graphics III

lab check-in

Announcements:

- for labtest#1 (this week), be prepared to:
  - create and draw a picture from a jpg file
  - create and draw a blank white canvas with a specified dimension
  - superimpose on top of this picture additional 2D Graphics
  - be able to draw these 4 shapes:
    - Line2D, Rectangle2D, RoundRectangle2D, Ellipse2D
    - in various sizes and in various locations
    - in filled and non-filled versions
    - with various stroke thicknesses
    - with various fills (other solid colours and/or gradient paint)

Goals/To do:

- How to create, copy, and delegate to aggregates
  - example aggregates: `Pixel`, `Picture`, `Graphics2D`
- Create, modify, and iterate over collections
- Implement traversal over a collection
- Implement search within a collection
- Use services of `Graphics2D` for drawing

Goals/To understand:

- recognize aggregates from their APIs
- characterize and distinguish between two traversal techniques
- distinguish between aliases, shallow copies, and deep copies of aggregrates
- understand the characteristics of the “current settings” graphical model

Today's Topics

- Aggregations vs Composition

Aggregation/Composition

Recap: The class `Date`

see `L05App01.java`

- a `Date` object can be used to represent a point in time
- the key attribute of any date object is its *time* value (a long)
  - the number of msec that have elapsed since **unix epoch**, Jan 01 00:00:00 UTC 1970
- accessor and mutator for Date object:
  - `getTime()`, `setTime(long)`

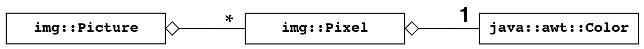
What is state? What is an aggregate?

- objects have *state*
  - primitive values do not have state, classes do not have state
  - *only objects have state*
- objects have zero or more **attributes** and attributes hold data
  - data may be primitive or non-primitive
- the object's state refers to **the specific values for all attributes of that object**
- if one or more attributes have non-primitive values*, then the object is an **aggregate**

* not including Strings

The `Picture` class is an aggregate

- If you want to change the Picture, you can use various accessors:
  - getPixels() : Pixel[]
  - getPixel(int, int) : Pixel
- once a reference to a Pixel object is obtained, you can use the mutators of the Pixel class to change the pixel
  - use the accessor of Picture + the mutator of Pixel
- conceptually, we are using **an accessor not a mutator** to change the Picture



The state of a `Picture` object

- `height : int`
- `width : int`
- a collection of pixels `: Pixels[]`
- a graphics2D object `: Graphics2D`
- `sourceFileName : String`
- `title : String`

Looking ahead…

- suppose we want to **observe** the `Picture` object
- whenever the state of the `Picture` object changes, some action should be triggered…

- we will introduce the idea of an ***observer*** **later** in the course

The `CreditCard` class

- The `CreditCard` class encapsulates a credit card and maintains information about it.
- Each card object has the following attributes:
  - card number : String
  - holder's name : String
  - issue date : Date
  - expiry date : Date
  - credit limit : double
  - balance owing : double

The `CreditCard` class

- Information about the card number:
  - 8 characters long, consisting of:
    - a 6-digit string
    - a dash, and
    - a MOD-9 check digit.
      - a digit such that the sum of all 6+1 = 7 digits will be a multiple of 9
      - it is added to detect possible transmission errors
  - the client of the constructor must specify the 6-digit string

The `CreditCard` class

```
type :: lib :: CreditCard
------------------------------------------------
CreditCard(int, String)
CreditCard(int, String, double)
CreditCard(int, String, double, Date)
+charge(double): boolean
+credit(double): void
+equals(Object): boolean
+getBalance(): double
+getExpiryDate() : Date
+getIssueDate() : Date
+getLimit() : double
+getName() : String
+getNumber() : String
+hashCode() : int
+isSimiliar() : boolean
+pay(double) : void
+setExpiryDate(Date) : boolean
+setLimit(double) : boolean
+toString(): String
```

The state of a `CreditCard` object

```
card number : String
card Name : String
issueDate : Date
expiryDate : Date
creditLimit : double
balance : double
```

The `CreditCard` class

```
type :: lib :: CreditCard
------------------------------------------------
CreditCard(int, String)
CreditCard(int, String, double)
CreditCard(int, String, double, Date)
+charge(double): boolean
+credit(double): void
+equals(Object): boolean
+getBalance(): double
+getExpiryDate() : Date
+getIssueDate() : Date
+getLimit() : double
+getName() : String
+getNumber() : String
+hashCode() : int
+isSimiliar() : boolean
+pay(double) : void
+setExpiryDate(Date) : boolean
+setLimit(double) : boolean
+toString(): String
```

- issue date defaults to moment of invocation,
- expiry date is issue date + 2 years
- credit limit defaults to $1000

- issue date defaults to moment of invocation,
- expiry date is issue date + 2 years
- limit is specified

- issue date defaults to moment of invocation,
- limit and expiry are specified

The `Picture` class vs the `CreditCard` class

- we used the accessor of `Picture` to mutate a `Picture` object
- can we use the accessor of `CreditCard` to mutate a `CreditCard` object?
- E.g.,
  - use accessor `getPixel(int, int)`, then mutate the `Pixel` object via `setColor(Color)`
  - use accessor `getIssueDate()`, then mutate the `Date` object via `setTime(long)`

Mutating a `CreditCard` object

suppose we want to change the expiry date to one day later…

- see `L05App02.java` for an approach that **does not** work
- we used the accessor of `CreditCard` to gain access to the `Date` object that corresponds to the expiry date attribute (or so we thought)
- we use the mutator of the `Date` object to add one day's worth of msec to the `Date` object's time attribute.
- <u>But this did not change the expiry date of the credit card.</u>
- Why not? Because the accessor of `CreditCard` returns a reference **to a copy** of the `Date` object, not a reference to **the actual** `Date` object.
  (compare to Picture, whose accessor returns references to the actual Pixel objects, not copies of the Pixel objects)
- When a class has this behaviour, we describe it as composition (see Ch 8)