CSE 1720

Lecture 7

Inheritance

Announcements:

- Lectures 7-10 assigned reading: Ch 9, JBA

Goals/To do:

- Good practices for the declaration and instantiation of objects within a class hierarchy
- Take advantage of polymorphism when desiging apps
- Create, modify, and iterate over a collection of Shapes; use services of `Graphics2D` for manipulating and/or operating upon the shape objects

Goals/To understand:

- understand a class in terms of its position within a hierarchy
- understand the `Object` class in terms of its position at the top of the class hierarchy
- recognize and understand subclass features from their APIs
- distinguish between early and late binding
- understand and distinguish among non-primitive types defined by: *classes*, *abstract classes* and *interfaces*.
- understand *generic collections*

Key Concepts

- We live in a world of many objects.
- Many apps require us to represent the world (albeit partially)
- To do this, we look to identify groups of objects
  - what is **common** among objects within a group?
    - similarity in terms of *attributes* and *methods.*
  - what sets objects **apart** within a group?
    - differences of *identity* and *state*
- We abstract the world in terms of classes and instances of these classes.
  - e.g., there are many object; a whole bunch of these objects are Cars. There are many actual cars, and these can be seen as instances of the class Car (each with a make, model, etc).

Key Concepts

- a class defines a new *non-primitive type*
- there are different types of classes
  - child classes, parent classes
  - "regular" classes, abstract classes, interface classes
- We can use these different types of classes to achieve *layered abstraction* in our apps

Key Concepts

- today: we will discuss a "regular" parent class with a child class
  - `CreditCard`, `RewardCard`
- later: an abstract parent class
  - `Arc2D`, child class `Arc2D.Double`, `Arc2D.Float`
  - `RectangularShape`, child classes: `Arc2D`, `Ellipse2D`, `Rectangle2D`, `RoundRectangle2D`
- later: an interface parent class
  - `Shape`, child class `RectangularShape`, `Line2D`, `CubicCurv2D`, `QuadCurv2D`

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

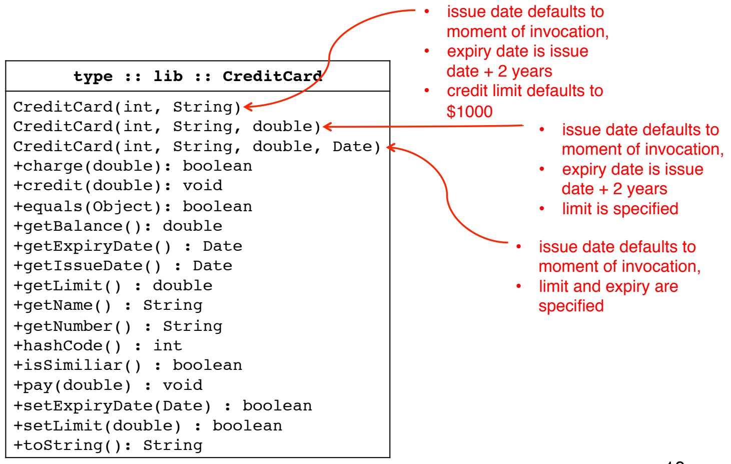
The `CreditCard` class

- 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 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

Now let's consider a specialized version of the credit card…

- A reward credit card is just like a credit card, with the addition of a points balance
- every purchase amount contributes towards the holder's points balance
- Every $20 worth of purchase results in 1 point.

Now let's look at RewardCard

- `RewardCard` IS-A `CreditCard`
- `RewardCard` is a specialization of `CreditCard`
- `RewardCard` is a child class of `CreditCard`
- `CreditCard` is a generalization of `RewardCard`

- how to spot specialization in the API:
  - look for `extends` in the class header

```
public class RewardCard
extends CreditCard
```

A series of sample apps

- `L07App01` demonstrates the existence of the *inherited* method `getName()`
- `L07App02` demonstrates the existence of the *overridden* method `toString();`
- `L07App03` demonstrates the existence of another *overridden* method
- `L07App04` demonstrates the existence of an *new* method in a child class (and shows how the child method cannot be used on an instance of the parent)
- `L07App05` demonstrates the existence of an *polymorphic* method
- `L07App06` demonstrates the need to manually cast an object to a child instance