CSE1720

Week 09, Lecture 15

Winter 2014 ◆ Tuesday, Mar 04, 2014

YORK UNIVERSITÉ UNIVERSITY

General Info

- Did you complete the lab exercises?
  - only 15 submissions as of this morning
- Did you read Chapter 10?
- Are your preparations underway for Term Test #2?
  - scheduled: Thu Mar 20
  - covers Chapter 9, Chapter 10, concepts from codebase
- Are your preparations underway for Lab Test #3
  - scheduled Thu Mar 13/Fri Mar 14
  - covers concepts from lab exercises

YORK UNIVERSITÉ UNIVERSITY

Example of polymorphism

```
import type.lib.CreditCard;
import type.lib.RewardCard;

public class L13Ex02 {

        public static void main(String[] args) {

                CreditCard tomsVisa;
                tomsVisa = new RewardCard(123458, "Tom Bentley");
                tomsVisa.charge(88);
                System.out.println(tomsVisa.toString());
        }
}
```

At compile time (during early binding), the variable `tomsVisa` resolves to `CreditCard` type. The compiler locates the method `charge(double)` within that class (binds to most specific), and records the signature `charge(double)` in the bytecode

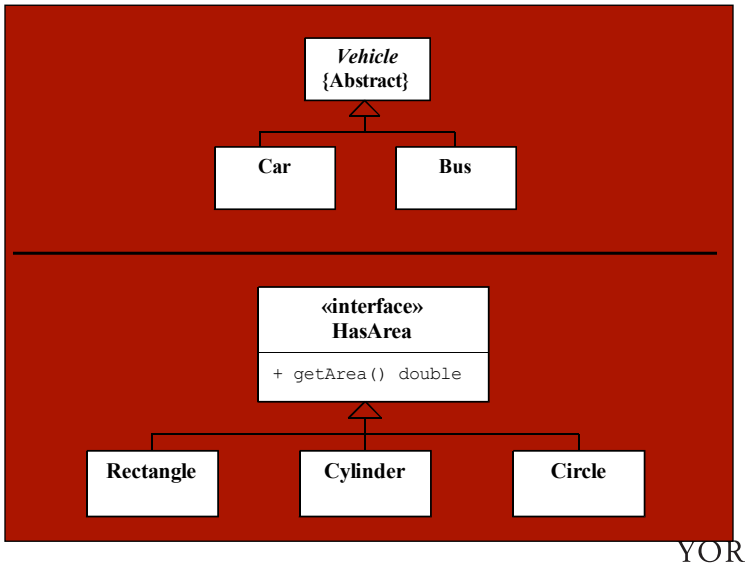
At runtime (during late binding), the variable `tomsVisa` resolves to the `RewardCard` type (since that is the type of the object in memory). The VM cues up the signature `charge(double)` and locates that method in the `RewardCard` class definition.

Is there a danger that such a method will not be found???

convince yourself that there is **no** such danger

3

9.2.4 Abstract Classes & Interfaces



4

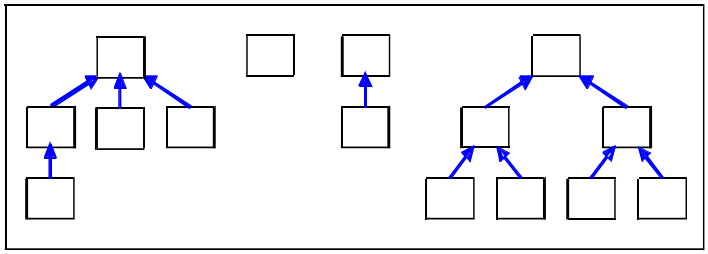
Abstract Classes & Interfaces, cont.

Key points to remember:

- **How to recognize an abstract class or an interface given its API or UML diagram.**
- **Both can be used as types for declarations.**
- **An abstract class cannot be instantiated. Instead, look for a concrete class C that extends it (or for a factory method that returns an instance of C).**
- **An interface class cannot be instantiated. Instead, look for a class C that implements it.**

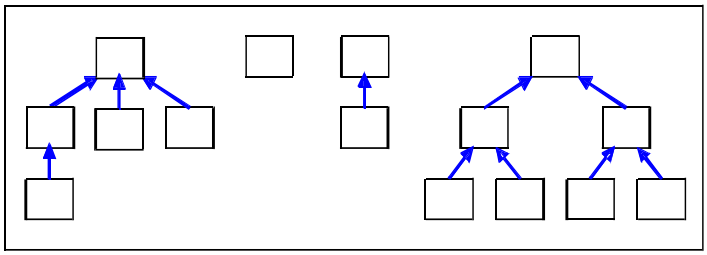
Example: the class `Calendar`.

9.3 Obligatory Inheritance



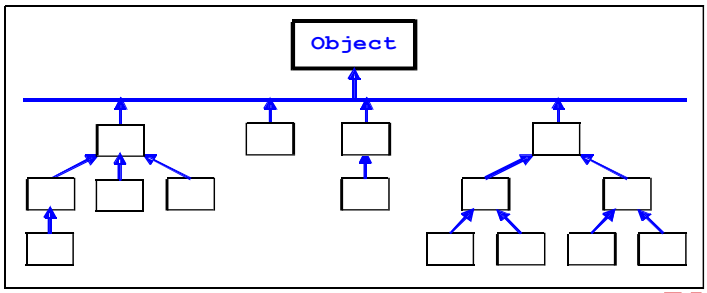
9.3 Obligatory Inheritance

Option #1



Option #2

Java uses this one.



9.3.1 The Object Class

Conclusion:

All classes have the features present in Object (unless they overrode them). They include:

- toString()
- equals()
- getClass()

9.3.2 Case Study: Object Serialization

Serialize = Write the state of an object to a stream

1. **Create an output stream connected to a file:**
   ```
   FileOutputStream fos;
   fos = new FileOutputStream(filename);
   ```
2. **Create an object output stream that feeds the file output stream:**
   ```
   ObjectOutputStream oos;
   oos = new ObjectOutputStream(fos);
   ```
3. **Serialize an object x:** `oos.writeObject(x);`
4. **Close the stream:** `oos.close();`

Object Serialization, cont.

De-serialize = Reconstruct a serialized object

```
FileInputStream fis;
fis = new FileInputStream(filename);

ObjectInputStream ois;
ois = new ObjectInputStream(fis);

x = (cast*) ois.readObject();

ois.close();
```

***The cast is needed because readObject returns an Object**

9.3.3 Generics

- **Components that take Object parameters are very flexible because they handle any type.**
- **But this flexibility thwarts all the benefits of strong typing (casts=potential runtime errors)**
- **The solution is a component that can take one specific type but that type is client-defined**
- **Such generic components provides flexibility and strong typing.**

Motivation

- You need to understand **the collection framework** in order to implement and to analyze any type of non-trivial application.
- The collection framework is specific to Java, but it relates to a more general concept in computer science.
  - Abstract data type (ADT) : mathematical models for certain types of data structures, used in order to describe and analyze abstract algorithms.
- The collection framework is just Java's implementation of certain ADTs.

About Abstract Data Types…

- Examples of ADTs:
  - Collection*, Deque, List*, Map*, Queue, Set*, Stack, Tree, … (among others)

*indicates an ADT implemented in the Java platform throught the Collections Framework

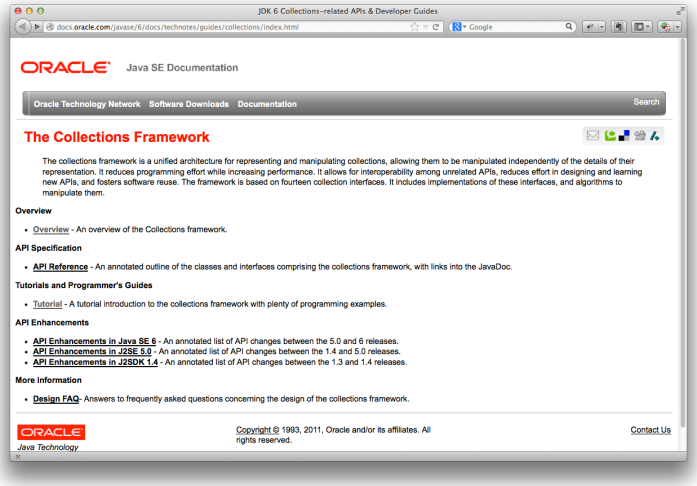
- an ADT is described solely in terms of:
  - the operations that may be performed on it (sound familiar?)
  - the mathematical constraints on the effects of the operations (e.g., insertion should require additional memory, etc)
- For most programming language, you can find implementations for most or all of the ADTs;
  - if you can't find an implementation, you will need to write it yourself!

13

YORK UNIVERSITÉ UNIVERSITY

Java SE Documentation for the Collections Framework

http://docs.oracle.com/javase/7/docs/technotes/guides/collections/index.html



14

YORK UNIVERSITÉ UNIVERSITY

Recap and Discussion

- We will discuss and distinguish among:
  - When should a **set** be used?
  - When should a **list** be used?
  - When should a **collection** be used?
  - When should a **map** be used?

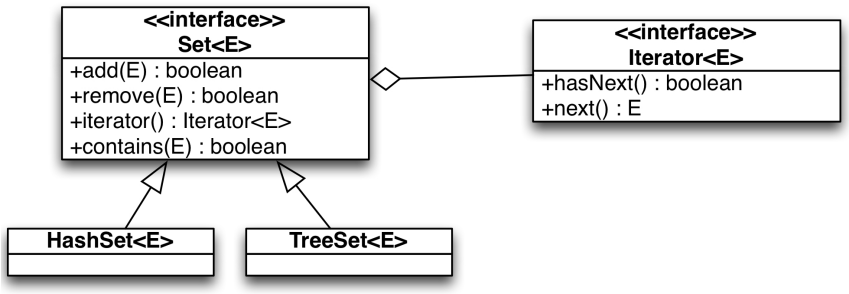
We will distinguish between:

- using of services for the purposes of *declaration*
- using of services for the purposes of *instantiation*

15

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface



16

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface

- The interface is `Set<E>`
- Implementing classes are `HashSet<E>` and `TreeSet<E>`

```
Set<String> s = new HashSet<String>();
Set<String> s = new TreeSet<String>();
```

Declaration

Instantiation

17

YORK UNIVERSITÉ UNIVERSITY

The `Set<E>` Interface

- The `Set` interface is "generic", which is indicated by the < and > in the interface name.
  - If you want to use the `Set` interface or the `HashSet` or `TreeSet` classes, you need to specify the type of the elements by writing it between < and >
  - By doing this, the client ensures:
    - No rogue element can be inserted
    - No casting is needed upon retrieval

18

YORK UNIVERSITÉ UNIVERSITY

HashSet<E> vs TreeSet<E>

Suppose your set contains 128 elements, ($\log_2$ 128=7)

- If you use a `HashSet<E>`, then
  - it will take 1 step to **add** an additional element
  - it will take 128 steps in the worst case to **remove** an element
  - it will take 128 steps in the worst case to **test whether a given element is found** within the set
- If you use a `TreeSet<E>`, then:
  - it will take 7 steps to **add** an additional element
  - it will take 7 steps to **remove** an additional element
  - it will take 7 steps to **test whether a given element is found** within the set

19

YORK UNIVERSITÉ UNIVERSITY

HashSet<E> vs TreeSet<E>

- If you use a `HashSet<E>`, then
  - the iterator will provide the elements in some sort of order that may or may not be sorted
- If you use a `TreeSet<E>`, then:
  - the iterator will provide the elements in a sorted order

*but wait – didn't we say that sets are **not sorted**?*

Yes, that's correct. The API doesn't require this, it just happens to be a kind of "bonus" of the TreeSet implementation

20

YORK UNIVERSITÉ UNIVERSITY

Pros and Cons…

Version #1

```
HashSet<String> s = new HashSet<String>();
TreeSet<String> s = new TreeSet<String>();
```

Version #2

```
Set<String> s = new HashSet<String>();
Set<String> s = new TreeSet<String>();
```

Discuss implication of versions #1 and #2

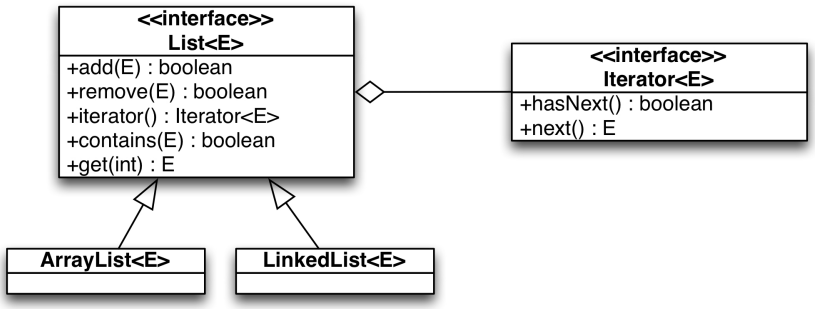
YORK UNIVERSITÉ UNIVERSITY

Best Practises

- *Declaration* as high up the hierarchy as possible
- *Instantiation* lower in the hierarchy

YORK UNIVERSITÉ UNIVERSITY

The `List<E>` Interface



23

YORK UNIVERSITÉ UNIVERSITY

`ArrayList<E>` vs `LinkedList<E>`

Suppose your list contains 128 elements ($\log_2 128=7$)

- If you use a `ArrayList<E>`, then
  - it will take 1 step to **get** an element
  - it may take up to 128 steps to **add** an additional element
  - it will take 128 steps to **remove** an element
- If you use a `LinkedList<E>`, then:
  - it will take 128 steps to **get** an element
  - it will take 1 step to **add** an additional element
  - it will take 7 steps to **remove** an additional element

`LinkedList<E>` is better if you need to add or remove elements

24

YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- A Map is a kind of *generalized* collection
- The elements are **pairs of objects**

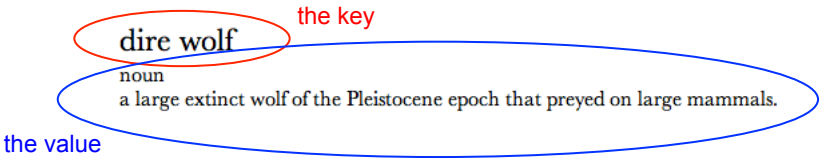
	Sets	Lists	Maps
elements are:	objects	objects	**pairs** of objects, consisting of a key and a value (key, value)
duplicates allowed?	no	yes	no, each pair must have a unique key
attempt to insert duplicate?	not inserted	inserted	"clobber" element that was already present

25

YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- a pair has two elements: the **key** and the **value**
- a dictionary is an example of a map
  - a dictionary consists of a list of pairs
  - the keys are sorted in lexicographic order



26

YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- When declaring a map, we need to specify:
  - the type of the keys
  - the type of the values

```
Map<String, String> theDictionary;
Map<String, Integer> theTally;
```

27

YORK UNIVERSITÉ UNIVERSITY

Discussion about Maps

- The basic operations
  - `put` a pair into the `Map`
  - `remove` a pair
  - given a key, `get` its corresponding value
  - iterate *over the keys*
  - iterate *over the values*

28

YORK UNIVERSITÉ UNIVERSITY

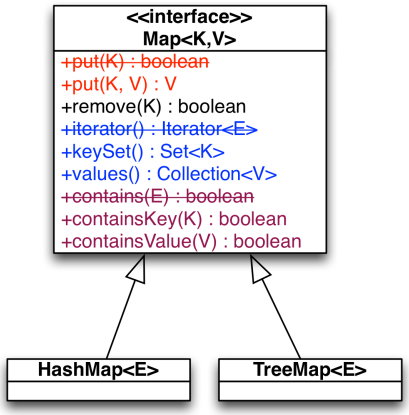
Issues

- `put` a pair into the `Map`
  - what if the key is already present?
- `remove` a pair
  - what if the key is not present?
- given a key, `get` its corresponding value
  - what if the key is not present?

29

YORK UNIVERSITÉ UNIVERSITY

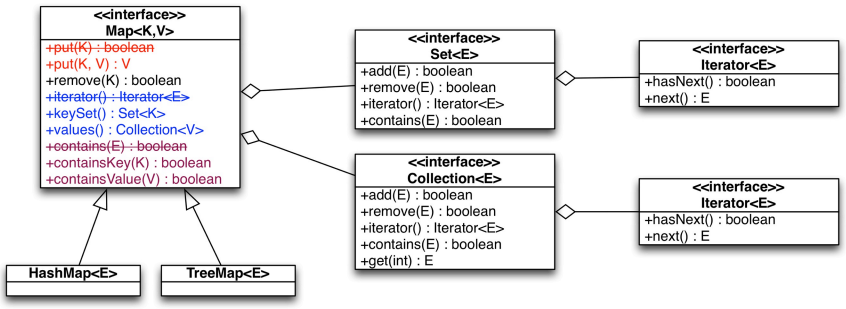
The `Map<K,V>` Interface



30

YORK UNIVERSITÉ UNIVERSITY

The `Map<K,V>` Interface



31

YORK UNIVERSITÉ UNIVERSITY

Instructions

- For each of the following design scenarios, state which *parameterized* type from the Collections framework is most appropriate and why.
  - For instance, if your answer is `Set<String>`, state why you chose `Set` for the collection and why you chose `String` for the elements of the collection.
  - If you think there is more than one correct answer, then make a decision, state your assumptions and describe the considerations in making your choice.

32

YORK UNIVERSITÉ UNIVERSITY

Instructions

- The possible choices are:
  - `Set<E>`
  - `List<E>`
  - `Map<K, V>`

  where:
  - `E` is the *type* of the element in the set/list, and
  - `K` and `V` are the *types* of the keys and values of the map, respectively.

  For the types `E`, `K`, and `V`, you may name class definitions that were used in the course or invent new class names.

33

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #1

- Goal: represent all of the songs a particular band intends to play at their next concert.

34

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #2

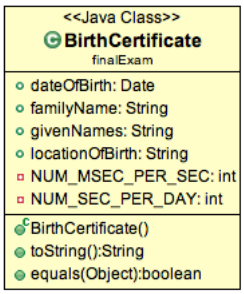
- Goal: record the *primary phone number* for each student in the university (which a student provides upon registration).
  - The goal in creating this collection is to provide it to a robo-calling service in order to issue a reminder message to everyone about the importance of vaccinations.
  - In order to not cause undue irritation, there should only be one robo-call per number.

35

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #3

- Goal: record the birth information of all of the students presently enrolled at York university.
  - You may use the services of the `BirthCertificate` class, which is shown in the UML diagram below.



36

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #4

- Goal: for every family name that is found among the students at York, record for each family name the birth certificate of the oldest student at York who has that family name.
  - You may use the `BirthCertificate` class from the previous Design Scenario

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #5

- Goal: record, for each day of the calendar, the number of people whose birthdate falls on that day.
  - You may use the `BirthCertificate` class from the previous Design Scenario
-

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #6

- Goal: represent all of the credit cards that are held by all of the students in the university.
  - The goal in creating this collection is so that the total actual debt and potential debt of the entire student population can be determined.
  - Assume for the moment that all privacy laws have been suspended.

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #7

- Goal: record all observations about the vehicles that drove west on Steeles between 9am and 10pm today between Keele and Jane.
  - The goal in creating this collection is so that it can be analyzed with respect to proportion of high-occupancy vehicles (more than three passengers) and low-occupancy vehicles (2 or fewer passengers).
  - Assume the class `VehicleObservation` is defined to encapsulate all of the attributes about a particular observation that was made of a particular vehicle (e.g., its license plate, its speed, the time and intersection of entering Steeles, the time and intersection of leaving Steeles).

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #8

- Goal: Building on design scenario #7, we wanted to create a lookup table so that, for any vehicle license plate, we can check the number of times it was observed travelling west on Steeles between Keele and Jane.

YORK UNIVERSITÉ UNIVERSITY

Design Scenario #9

- Goal: Building on design scenario #7, suppose we wanted to create a lookup table so that for any vehicle license plate, we could determine all of its vehicle observations on Steeles between Keele and Jane.

YORK UNIVERSITÉ UNIVERSITY