CSE1720

Week 08, Lecture 13

Winter 2014 ◆ Thursday, Feb 27, 2014

YORK UNIVERSITÉ UNIVERSITY

General Info

- Continuation of Chapter 9
- Read Chapter 10 for next week

YORK UNIVERSITÉ UNIVERSITY

9.1 What is Inheritance?

For each pair, determine the relationship if any

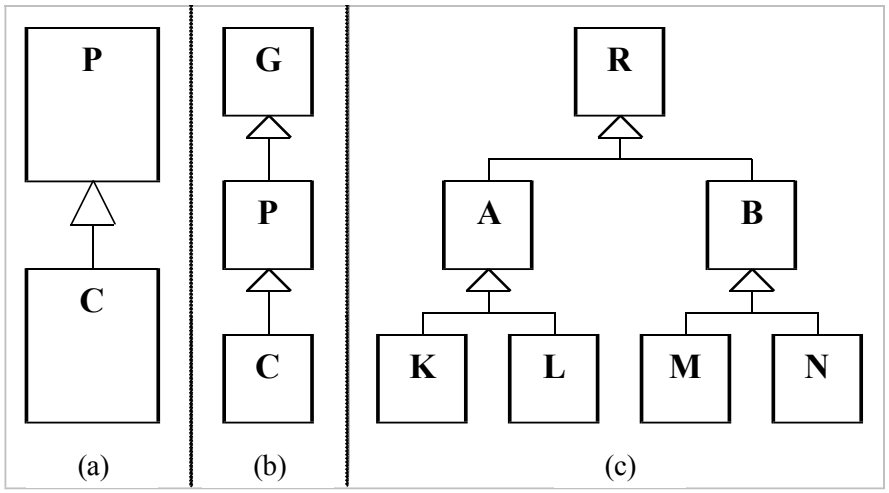
- Camera, Film
- Vehicle, Car
- Library, Book
- Animal, Dog
- Car, Tree

When can you say that the 2nd is a subclass of the 1st

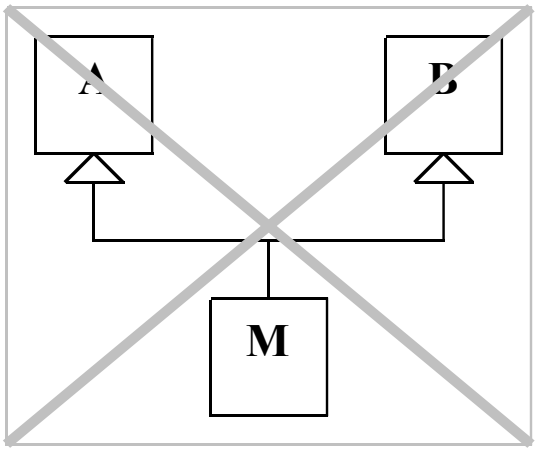
9.1.1 Definition and Terminology

- The API of a class C may indicate that it extends some other class P
- Every feature of P is in C
- C inherits from P.
- Child-Parent, Subclass-Superclass
- Inheritance = is-a = Specialization
- Inheritance chain, hierarchy (root, descendents, ascendant)

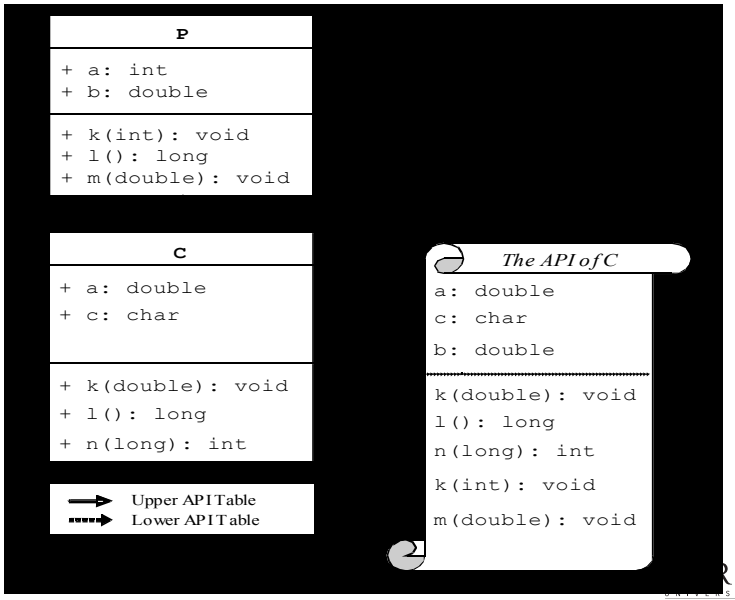
UML



No Multiple Inheritance



9.1.2 The Subclass API



Feature Classification

- Inherited from parent
  Lower table
- Added as new by child
  Upper table
- Overriding by child (same signature)
  Upper table
- Shadowing by child (same name)
  Upper table

Note: a child cannot override with a diff return!

Feature Count

Is this correct?

- **x = #of methods in parent's UML**
- **y = #of methods in child's UML**
- **The child's API shows x + y methods (upper plus lower)**

Repeat for fields.

Example

- `Graphics2D` is a subclass of `Graphics`
  - `Graphics` was the first implementation, in later versions, `Graphics2D` was released as a more sophisticated implementation
- The method `drawString(String, int, int)` is defined in Graphics
- The method `drawString(String, int, int)` is defined in Graphics2D
- Thus, the method is **overridden** in `Graphics2D`

Example

- There are 6 different `drawImage` methods in `Graphics`
- There are 2 additional `drawImage` methods in Graphics2D.
  - These two have different signatures from the `drawImage` methods in `Graphics`
- Thus, the method is **shadowed** in `Graphics2D`
- How many `drawImage` methods are available to a `Graphics2D` instance?

YORK UNIVERSITÉ UNIVERSITY

9.2.1 The Substitutability Principle

When a parent is expected, a child is accepted

- **Similar to substituting "man" or "woman" in The fare is $5 per person**
- **Similar to automatic promotion of primitives.**
- **Compiler uses it in:**
  - LHR / RHS of an assignment
  - Parameter passing
  - catch blocks, throws statements

YORK UNIVERSITÉ UNIVERSITY

9.2.2 Early & Late Binding

How do you bind: r.m(…) ?

Early and Late Binding

- We have the statement:
  `r.m(…);`
- The compiler performs **early binding**
  - this means to find **the target** to which **an identifier** resolves
  - all **identifiers** must resolve (must correspond to a definition, somewhere)
- Then we have the bytecode that is generated from `r.m(…);`
  - The VM performs **late binding**

Early Binding, Binding with Most Specific

- To bind `r.m(…)` here's what the compiler does:
  - determines the declared type of `r`. Suppose it is C
    - If it cannot be found, issue r cannot be resolved to a variable.
  - locates the signature `m(…)` in C
    - If more than one such `m` is found, it binds with the "most specific" one.
    - Record the signature S of the found method
    - If no such `m` can be found, then issue Cannot Resolve Symbol error.

YORK UNIVERSITÉ UNIVERSITY

Late Binding

- To bind the bytecode from `r.m(…)` bytecode, here's what the VM does:
  - cue up the signature S that was recorded in the bytecode
  - determine the referent of `r`
    - some object will be located in run-time memory
    - if there is no such object, issue Null Pointer Exception
  - determine the type of `r`. Suppose the class is Q
    - look for a method with the signature S in Q
    - If Q is the same as C (same as declared type), then the signature S certainly will be found
    - If Q is the NOT the same as C, then Q **MUST** be a sub-type of C. The signature S will be found, since it will be either **inherited** or will have been **overridden**.

YORK UNIVERSITÉ UNIVERSITY

9.2.3 Polymorphism

- An invocation of an overridden method is **polymorphic**
- The meaning changes.
    - At compilation, the binding will be to the parent's method.
    - At runtime, the binding will be to the child class's overriding method
- Polymorphism leads to elegant programs.
  No if statements and no redundancies.

YORK UNIVERSITÉ UNIVERSITY

9.2.3 Polymorphism

- What if I have something like this:
  `P r = new C();`
- What if I want to invoke a method on `r` that is present only in the child class `C`, such as `n()`?
- polymorphism will not be available
- for compilation, must use a cast (down the chain)
  `(C) r.n()`
- But this can lead to a runtime exception.
  To avert, use instanceof before casting

```
if (r instanceof C)
   (C) r.n();
```


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

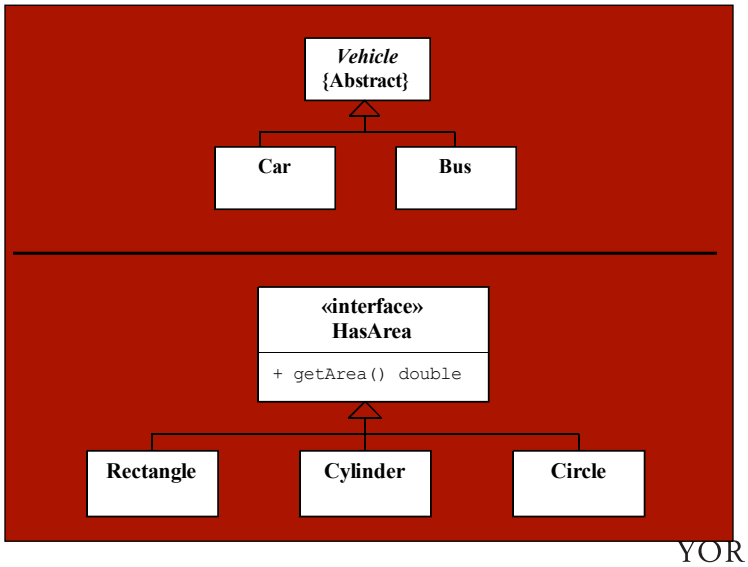
At runtime (during late binding), the variable `tomsVisa` resolves to the `RewardCard` type (since that is the type of the object in memory). The VM cues up the signature `charge(double)` and locates that method in the `RewardCard` class definition.

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

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

19

9.2.4 Abstract Classes & Interfaces



20

 Java By Abstraction

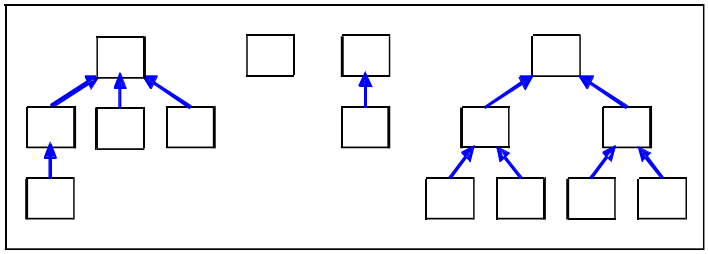
Abstract Classes & Interfaces, cont.

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.**

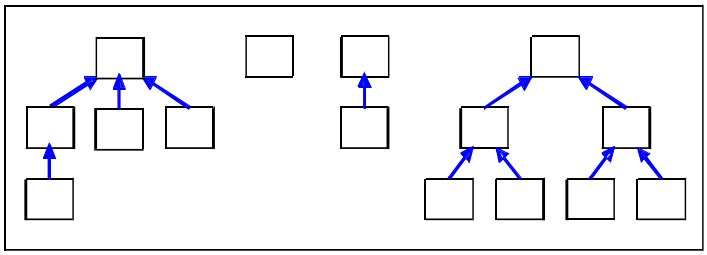
Example: the class `Calendar`.

9.3 Obligatory Inheritance



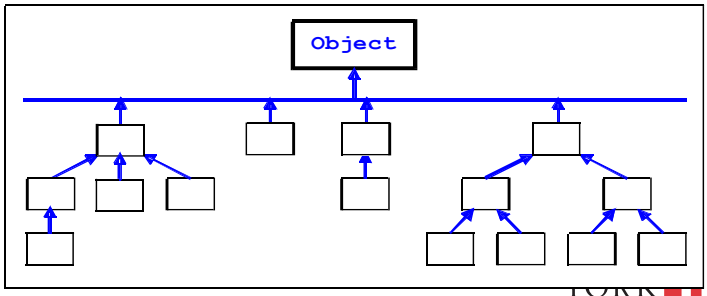
9.3 Obligatory Inheritance

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.