

Chapter 9

Inheritance

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 9-1

Outline

- 9.1 What is Inheritance?
 - 9.1.1 Definition and Terminology
 - 9.1.2 The Subclass API
 - 9.1.3 Case Study: Credit Cards
- 9.2 Working with Inheritance Hierarchies
 - 9.2.1 The Substitutability Principle
 - 9.2.2 Early and Late Binding
 - 9.2.3 Polymorphism
 - 9.2.4 Abstract Classes and Interfaces
 - 9.2.5 Case Study: Revisiting Streams
- 9.3 Obligatory Inheritance
 - 9.3.1 The Object Class
 - 9.3.2 Case Study: Object Serialization
 - 9.3.3 Generics

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 9-2

9.1 What is Inheritance?

For each pair, determine the relationship if any

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

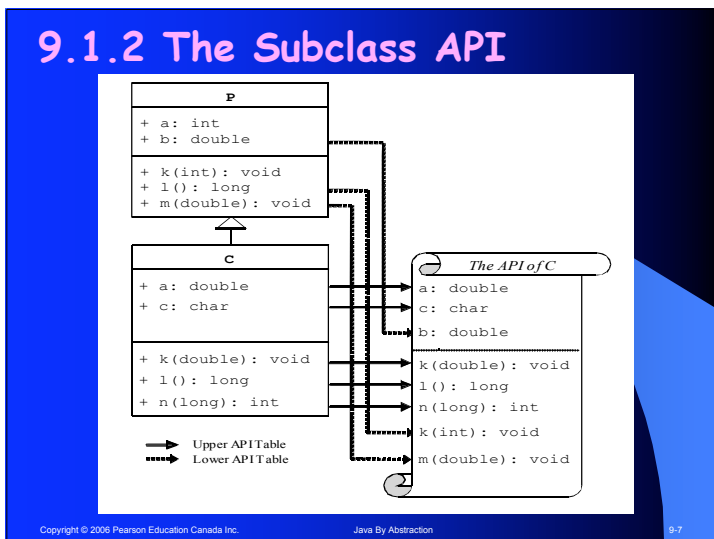
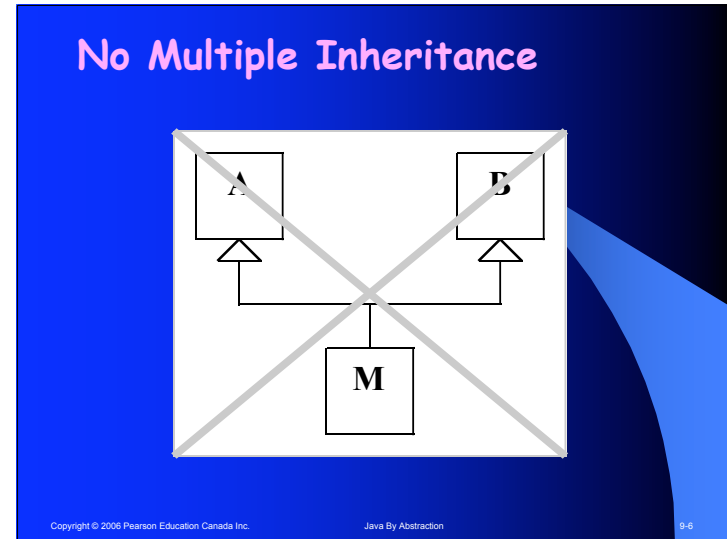
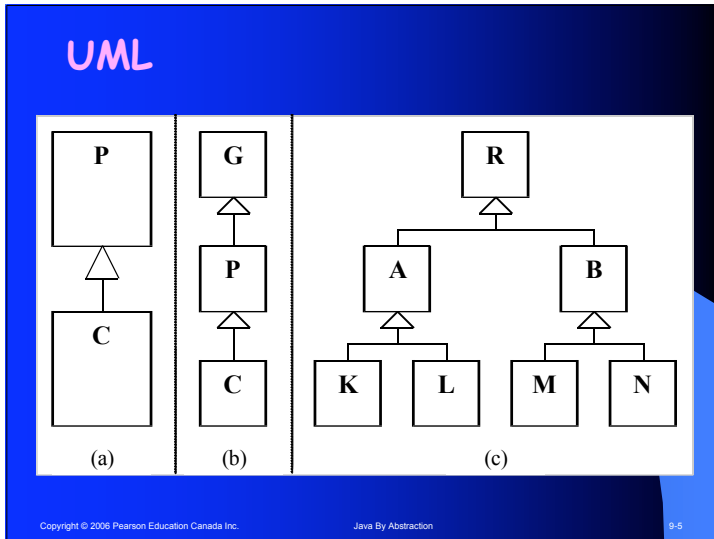
When can you say that the 2nd is a subclass of the 1st

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 9-3

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)

Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 9-4



- ### 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!**
- Copyright © 2006 Pearson Education Canada Inc. Java By Abstraction 9-8

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.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-9

9.1.3 Case Study: CC-RW

Examine the API of CreditCard

- Issue a card #9 to Adam
- Charge \$500 on it
- Pay back \$300
- How many dollars does Adam owe?

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-10

RewardCard

- Issue a reward card #9 to Adam
- Charge \$500 on it
- Pay back \$300
- How many does Adam owe?
- How many reward points does he have?

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-11

Case Study, cont.

Examine the API of RewardCard

- Is the constructor inherited?
- How many fields does it have?
- How many methods does it have?
- Provide a rationale as to why certain methods were **overridden**, or **added**.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-12

9.2 Working with Hierarchies

- Inheritance is no problem as long as client deals with **one class at a time**
- Just watch out for multiple tables in the API of that class
- What if the client uses **several sub-classes** on a chain?

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-13

Example

Write a program that prompts the user for a card type and then instantiate the desired card and charge \$250 on it.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-14

Example

```
output.println("Ordinary or Reward [O/R]?");
char type = input.nextChar();
if (type == 'O')
{
    CreditCard card;
    card = new CreditCard(9, "Adam");
} else
{
    RewardCard card;
    card = new RewardCard(9, "Adam");
}
// charge the card
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-15

Example

```
output.println("Ordinary or Reward [O/R]?");
char type = input.nextChar();
if (type == 'O')
{
    CreditCard card;
    card = new CreditCard(9, "Adam");
} else
{
    RewardCard card;
    card = new RewardCard(9, "Adam");
}
// charge the card But it is out of scope here!
```

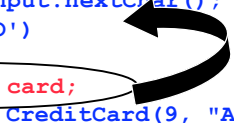
Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-16

Example

```
output.println("Ordinary or Reward [O/R]?");
char type = input.nextChar();
if (type == 'O')
{
  CreditCard card;
  card = new CreditCard(9, "Adam");
} else
{
  RewardCard card;
  card = new RewardCard(9, "Adam");
}
// charge the card
```



Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-17

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 primitive's.
- Compiler uses it in:
 - LHR / RHS of an assignment
 - Parameter passing

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-18

Substitutability Examples

Assigning RHS to LHS:

```
CreditCard card = new RewardCard(...);
```

Passing parameters:

```
CreditCard cc = new CreditCard(...);
RewardCard rc = new RewardCard(...);
if (cc.isSimilar(rc))
{
  ...
}
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-19

9.2.2 Early & Late Binding

How do you bind: `r.m(...)` ?

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-20

9.2.2 Early & Late Binding

How do you bind: `r.m(...)` ?

1. Search for `m(...)` in the **declared class** of `r`
2. If more than one, pick `S`, the **most specific**
3. If above failed, issue compile-time error

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-21

9.2.2 Early & Late Binding

How do you bind: `r.m(...)` ?

1. Search for `m(...)` in the **declared class** of `r`
2. If more than one, pick `S`, the **most specific**
3. If above failed, issue compile-time error

This is **early binding**. It is done at compile time and culminates in an error or a signature `S`.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-22

9.2.2 Early & Late Binding

How do you bind: `r.m(...)` ?

This is **late binding**. It is done at runtime and culminates in an error or a binding.

1. If `r` is null, issue runtime error (NullPointerException)
2. Search for `S` in **actual class** of `r` (the object)
3. You will find it ... guaranteed!

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-23

9.2.2 Early & Late Binding

How do you bind: `r.m(...)` ?

1. Search for `m(...)` in the **declared class** of `r`
2. If more than one, pick `S`, the **most specific**
3. If above failed, issue compile-time error

1. If `r` is null, issue runtime error (NullPointerException)
2. Search for `S` in **actual class** of `r` (the object)
3. You will find it!

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-24

Example

Bind all invocations:

```
CreditCard c1 = new RewardCard(9, "Jim");  
CreditCard c2 = new RewardCard(9, "Eve");  
  
c1.charge(500);  
c1.pay(500);  
  
output.println(c1.isSimilar(c2));
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-25

9.2.3 Polymorphism

- An invocation of an overridden method, e.g. `r.charge(500)`, is polymorphic
- The meaning changes (during late binding) based on the actual object type
- Polymorphism leads to elegant programs. No if statements and no redundancies.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-26

9.2.3 Polymorphism

- For methods that are only present in the child, polymorphism cannot be used.
- Must have a cast (down the chain)
- In such cases, use `instanceof` before casting

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-27

Example 1

Given that `card` is declared as `CreditCard`, find its point balance if applicable.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-28

Example 1

Given that `card` is declared as `CreditCard`, find its point balance if applicable.

First attempt:

```
if (card instanceof RewsrdCard)
{
    output.println(card.getPointBalance());
}
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-29

Example 1

Given that `card` is declared as `CreditCard`, find its point balance if applicable.

Correct solution:

```
if (card instanceof RewsrdCard)
{
    RewardCard rc = (RewardCard) card;
    output.println(rc.getPointBalance());
}
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-30

Example 2

Predict the output:

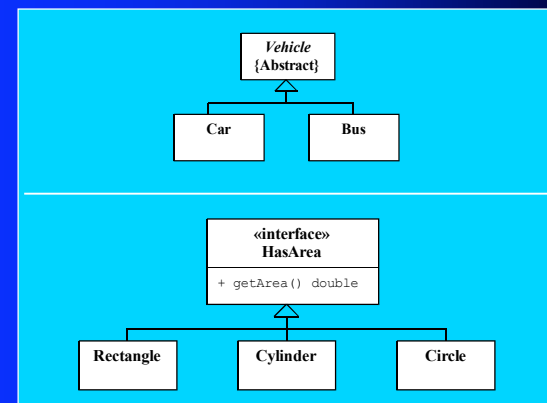
```
CreditCard c1 = new RewardCard(9, "Adam");
CreditCard c2 = new RewardCard(9, "Adam");
c1.charge(100);
c1.pay(100);
print(c1.isSimilar(c2));
print(c1.isSimilar((RewardCard) c2));
print(((RewardCard) c1).isSimilar(c2));
print(((RewardCard) c1).isSimilar((RewardCard) c2));
```

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-31

9.2.4 Abstract Classes & Interfaces



Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-32

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.

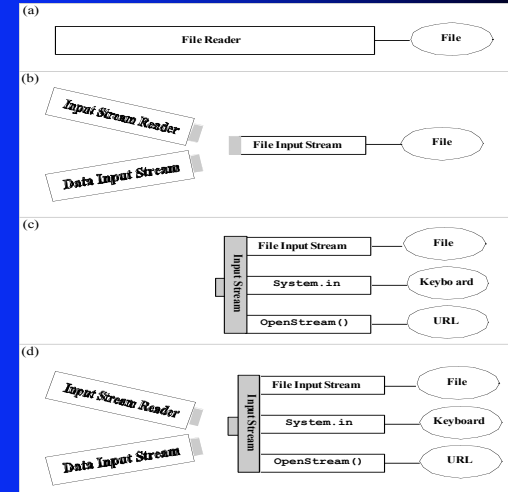
Example: create an instance of `Calendar`.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-33

9.2.5 Revisiting Streams

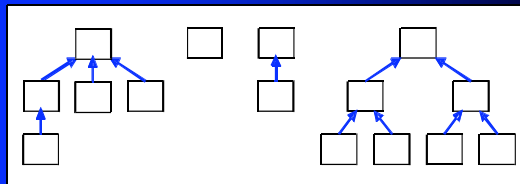


Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-34

9.3 Obligatory Inheritance



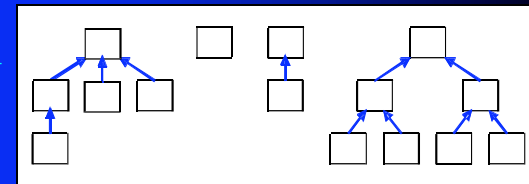
Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-35

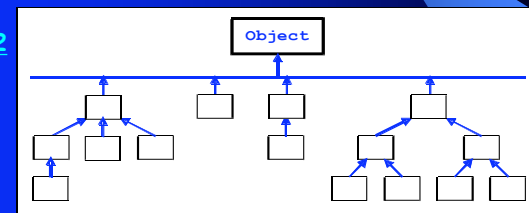
9.3 Obligatory Inheritance

Option #1



Option #2

Java uses this one.



Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-36

9.3.1 The Object Class

Conclusion:

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

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

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-37

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();`

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-38

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

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-39

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.

Copyright © 2006 Pearson Education Canada Inc.

Java By Abstraction

9-40