Drop deadline

Friday March 7

Until this date you can drop the course without getting a grade for it and, hence, it will not affect your gpa.

Petitions to drop this course after the drop deadline will not be granted automatically.

Aggregation

Combine simple data into more complex data.

Inheritance

Add some simple data to already existing complex data.

For the resulting data, add new operations (mainly to handle the added simple data) and possibly redefine some of the operations of the complex data.

Inheritance

Inheritance was invented in 1967 for the object-oriented programming language Simula.

"Inheritance is an object-oriented technique that allows you to re-use code across related objects in your applications."
Source: www.objectorientedcoldfusion.org/wiki/Inheritance

"Item 14: Favor composition (aggregation) over inheritance."
Source: Joshua Bloch. *Effective Java: Programming Language Guide*. Addison-Wesley. 2001.

Inheritance

Definition

Inheritance is a binary relation on classes. The pair (C, P) of classes is in the inheritance relation if the API of the class C (child) contains

```
class C extends P
```

The API of the class P (parent) may (but does not have to) contain

```
Direct Known Subclasses: C
```

The inheritance relation is also known as the is-a relation. Instead of saying that (C, P) is in the inheritance relation, we often simply say that C is-a P.

Inheritance

Example

`RewardCard` is-a `CreditCard`

`CEStudent` is-a `Student`

`ITStudent` is-a `Student`

`SEStudent` is-a `Student`

Inheritance

Definition

P is a superclass of C if C is-a P.

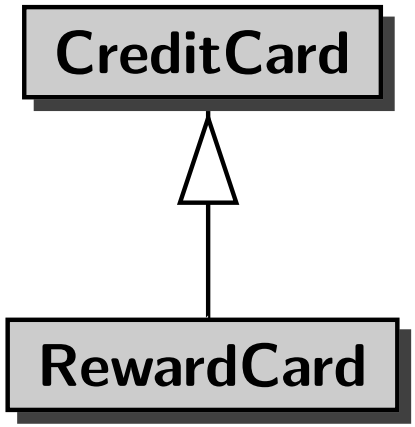
C is a subclass of P if C is-a P.

Example

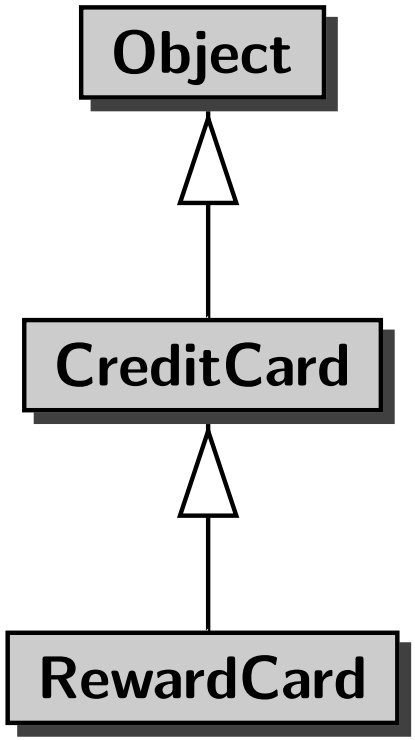
`Student` is a superclass of `CEStudent`

`RewardCard` is a subclass of `CreditCard`

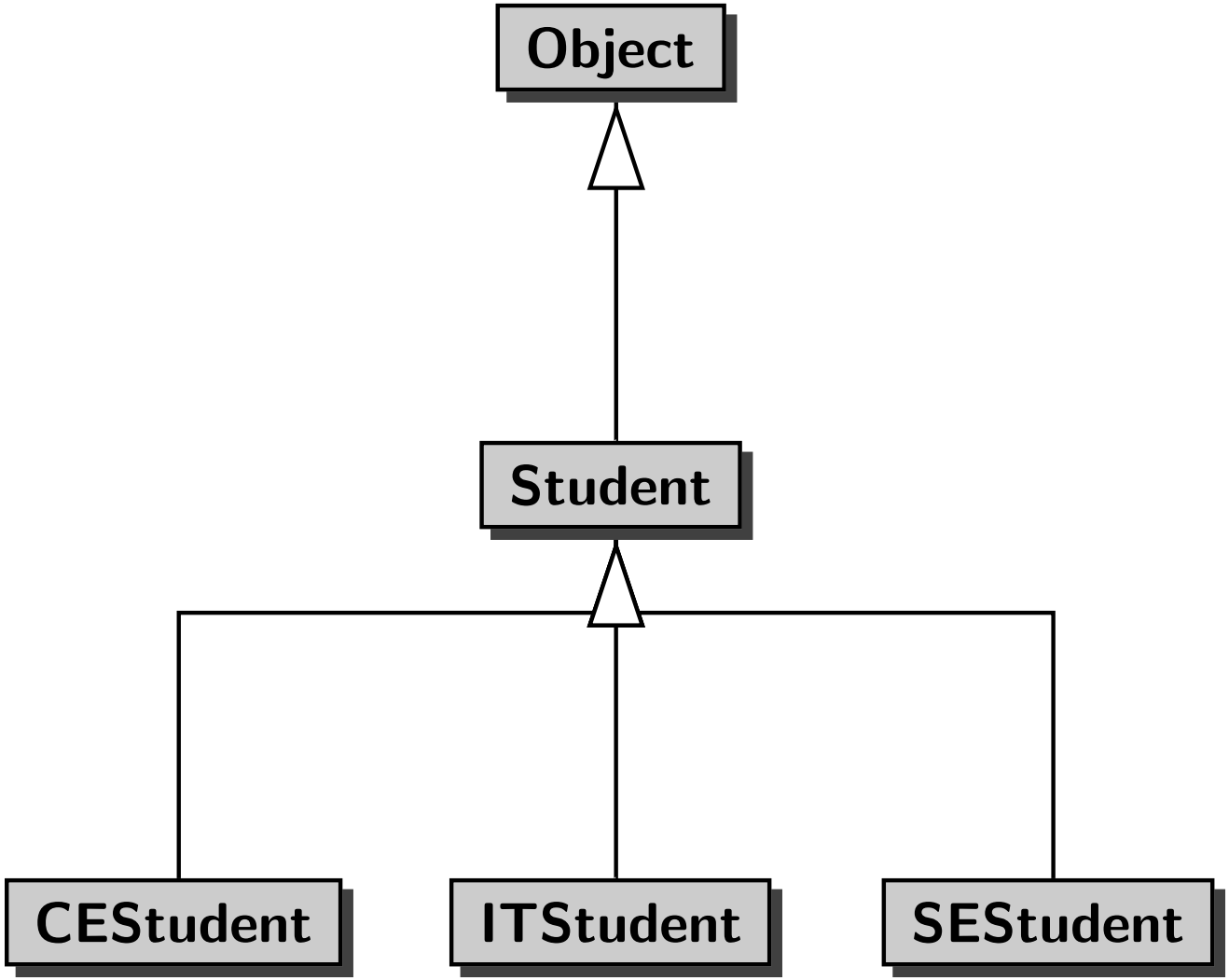
UML Diagrams



UML Diagrams



UML Diagrams



Inheritance

Definition

A programming language supports *single inheritance* if each class has at most one superclass.

A programming language supports *multiple inheritance* if each class may have multiple superclasses.

Example

Java supports single inheritance.

Eiffel supports multiple inheritance.

A Class Consists of

- constructors,
- attributes, and
- methods.

Constructors

Constructors are not inherited from the superclass.

In a Well-Designed Class ...

- ... all non-final attributes are private (page 77).

We will restrict ourselves to such well-designed classes.

Hence, we assume that

- all public attributes are final.

Attributes

All public non-static final attributes are inherited from the superclass.

For a well-designed class, these are the only non-static attributes of which a client is aware. The other non-static attributes are relevant to the implementer and are considered in CSE 1030.

Static attributes are not inherited. They can be accessed via the superclass (name).

Attributes

Question

Assume that the class P has a public non-static final attribute named a. Can its subclass C also contain a public non-static final attribute named a?

Attributes

Question

Assume that the class P has a public non-static final attribute named a. Can its subclass C also contain a public non-static final attribute named a?

Answer

Yes. In that case, the attribute a of the subclass C is said to *shadow* the attribute a of the superclass P.

However, why would one ever introduce two different constants with the same name? In well-designed classes, such a situation never arises.

Methods

All public non-static methods are inherited from the superclass.

The private non-static methods are relevant to the implementer and are considered in CSE1030.

Static methods are not inherited. They can be invoked via the superclass (name).

For example, the public non-static method `getBalance` of the class `CreditCard` is inherited by the class `RewardCard`.

Question

Assume that the class P has a public non-static method with signature (method name and parameter types) s. Can its subclass C also contain a public non-static method with signature s?

Question

Assume that the class P has a public non-static method with signature (method name and parameter types) s. Can its subclass C also contain a public non-static method with signature s?

Answer

Yes. In that case, the method with signature s of the subclass C is said to *override* the method with signature s of the superclass P.

Methods

We can distinguish between

- inherited methods
- overridden methods
- new methods

The substitutability principle

```
final int NUMBER = 3576836;
final String NAME = "Franck";
CreditCard card = new RewardCard(NUMBER, NAME);
output.println(card);
```

The assignment `CreditCard card = new RewardCard(NUMBER, NAME)` is valid, since a `RewardCard` is-a `CreditCard`.

Although the signature of the `println` method is `println(Object)`, the method call `output.println(card)` is valid, since a `CreditCard` is-a `Object`.

Early binding

When the compiler encounters the invocation

$\texttt{r.m}(a_1, \ldots, a_n)$

it performs early binding. It consists of the following three steps.

1. Determine the declared type of the object reference `r`: class `C`.
2. Find compatible methods `m` in class `C`.
3. Select the most specific compatible method $\texttt{m}(t_1, \ldots, t_n)$ in class `C`.

The invocation $\texttt{r.m}(a_1, \ldots, a_n)$ is bound to method $\texttt{m}(t_1, \ldots, t_n)$ of class `C`.

Early binding

Question

Consider the following snippet.

```
int i = 23;
double d = 3.0;
output.println(i);
output.println(d);
output.printf(i);
```

For each of the three method calls, determine the method to which it is bound.

Early binding and inheritance

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `println(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Early binding and inheritance

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `println(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Answer

`println(double)` of `PrintStream`.

Early binding and inheritance

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `println(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Answer

`println(double)` of `PrintStream`.

Question

Is this the method we want to invoke?

Early binding and inheritance

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `println(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Answer

`println(double)` of `PrintStream`.

Question

Is this the method we want to invoke?

Answer

No, we want `println(double)` of `MyPrintStream`.

Early and late binding

early binding	compiler	javac
late binding	virtual machine	java

Late binding

Assume that the compiler binds the invocation

`r.m(`$a_1, \ldots, a_n$`)`

to method `m(`$t_1, \ldots, t_n$`)` of class `C`.

When the virtual machine encounters the invocation

`r.m(`$a_1, \ldots, a_n$`)`

it performs late binding. It consists of the following step.

- Determine the actual type of the object reference `r`: class `C'`.

The invocation `r.m(`$a_1, \ldots, a_n$`)` is bound to method `m(`$t_1, \ldots, t_n$`)` of class `C'`.

Note that the signature does not change.

Late binding

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `print(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Late binding

Question

The class `MyPrintStream` extends the class `PrintStream`. The former class overrides the method `print(double)`. Consider the following snippet.

```
PrintStream output = new MyPrintStream(...);
output.println(1.0);
```

To which method is the method call bound?

Answer

`println(double)` of `MyPrintStream`.

Late binding

Question

Consider the following snippet.

```
CreditCard c1 = new CreditCard(...);
CreditCard c2 = new RewardCard(...);
RewardCard c3 = new RewardCard(...);
```

Determine the early and late binding of

c_i`.isSimilar(`c_j`)`

Credit card collection

Problem

Create a random collection of credit cards (use the `GlobalCredit` class) and print each card on a separate line.

Polymorphism

The `toString` method is said to be polymorphic, that is, it has multiple forms.

Credit card collection

Problem

Create a random collection of credit cards (use the `GlobalCredit` class) and print the total balance of all cards combined.

Credit card collection

Problem

Create a random collection of credit cards (use the `GlobalCredit` class) and print the total point balance of all reward cards combined.

instanceof

The Boolean expression

`r instanceof C`

evaluates to true if `r` is not `null` and its type is `C` or any of its descendants.

Casting: at compile time

Assume that the declared type of the reference `r` is `C`.

- Then `(C')r` gives rise to a compile time error if `C'` is neither a descendant nor an ancestor of `C`.
- If `(C')r` does not give rise to a compile time error, then its declared type is `C'`.

Casting: at compile time

Question

Assume that the declared type of reference `card` is `CreditCard`. Which of the following gives rise to a compile time error?

1. `(RewardCard)card`
2. `(CreditCard)card`
3. `(Object)card`
4. `(Integer)card`

Casting: at compile time

Question

Assume that the declared type of reference `card` is `CreditCard`. Which of the following gives rise to a compile time error?

1. `(RewardCard)card`
2. `(CreditCard)card`
3. `(Object)card`
4. `(Integer)card`

Answer

4.

Casting: at run time

`(C')r` gives rise to a run time error if the actual type of `r` is not a descendant of `C'`.

Casting: at run time

Question

Assume that the actual type of reference `card` is `CreditCard`. Which of the following gives rise to a run time error?

1. `(RewardCard)card`
2. `(CreditCard)card`
3. `(Object)card`

Casting: at run time

Question

Assume that the actual type of reference `card` is `CreditCard`. Which of the following gives rise to a run time error?

1. `(RewardCard)card`
2. `(CreditCard)card`
3. `(Object)card`

Answer

1.

To do

- Study Section 9.1 of the textbook.