Copyright Warning

COMMONWEALTH OF AUSTRALIA

Copyright Regulations 1969

WARNING

This material has been reproduced and communicated to you by or on behalf of the University of Sydney pursuant to Part VB of the Copyright Act 1968 (**the Act**).

The material in this communication may be subject to copyright under the Act. Any further copying or communication of this material by you may be the subject of copyright protection under the Act.

Do not remove this notice.

Copyright © ViSLAB, The University of Sydney

Object Oriented Design (OO != Real World)

Week 2-1

Copyright © ViSLAB, The University of Sydney

Don’t try to understand with just OO concept.

- OO == (?)
  - “Technique to represent the real-world in Software as is”
- You would observe some inconsistencies between real-wold and soft-representation.
  - Design pattern, Framework, etc....hard to map them to the real-world

Copyright © ViSLAB, The University of Sydney

Don’t try to understand with just OO concept.

- Just like OOT started in designing OOPL,
- don’t just learn OO concept...make sure do OOPL as well to see the implication on implementation.
  - some concepts might not directly translate to software objects/classes.

Copyright © ViSLAB, The University of Sydney

Look OO (vs Real World)

- Class
- Polymorphism
- Inheritance

Copyright © ViSLAB, The University of Sydney

Class == Kind
Instance == concrete object

- Class : The basic structure of OO
  - category of things
- Instance (object) :
  - an example or single occurrence of something

Copyright © ViSLAB, The University of Sydney

Class vs. Instance (examples)

- Class : Instances.
  - country: Australia, China, USA, ...
  - OS: Windows, Linux, OS X, ...
  - academics: lecturer, Assis/Prof, Assoc/Prof, Prof
  - ...

Copyright © ViSLAB, The University of Sydney

Class first

- In OOP, you first “define” a class.
- From the defined class, an instance (instances) are “created”
- Interaction between the instances achieves a task.
  - Ppl might say...”that’s why software objects in OO work just like real world objects (people, things)”.

Copyright © ViSLAB, The University of Sydney

Example (Java)

```
    // define Brother
public class Brother {
    private String name;    // name

    Brother(String name) {  // constructor (executed at the time of instance creation)
        this.name = name;
    }

    String cry() {          // make him cry.
        return "*sob*";
    }

    public static void main(String[] args) {
        Brother john = new Brother("John");
        Brother jack = new Brother("Jack");

        // make them cry
        System.out.println(john.cry());
        System.out.println(jack.cry());

        return;
    }
}
```

Anything odd?

Copyright © ViSLAB, The University of Sydney

Example (C++)

```
    #include <string>
#include <iostream>
using namespace std;

// define Brother
class Brother {
private:
    string name;        // name

public:
    Brother(string n) {  // constructor (executed at the time of instance creation)
        name = n;
    }

    string cry() {    // make him cry.
        return "*sob*";
    }
};

int main() {
    Brother john("John");
    Brother jack("Jack");

    // make them cry
    cout << john.cry() << endl;
    cout << jack.cry() << endl;

        return 0;
}
```

Anything odd?

Copyright © ViSLAB, The University of Sydney

Polymorphism (common way of messaging)

- the occurrence of something in different forms
- a mechanism to use a common messaging
  - to objects of similar (slightly different) classes.
  - without being conscious about each classes.

Copyright © ViSLAB, The University of Sydney

Example

- “Brother” class... you can make him cry.
- If you have “Dog” class, you can also make him cry.

Copyright © ViSLAB, The University of Sydney

```
#include <string>
#include <iostream>
using namespace std;

class Animal {
private:
    string name;
public:
    Animal(string name):name(name) {}
    virtual string cry() {}
};

// define Brother
class Brother : public Animal {
public:
    Brother(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "*sob*";
    }
};

// define Dog
class Dog : public Animal {
public:
    Dog(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "bowwow";
    }
};

int main() {
    Brother john("John");
    Dog jack("Jack");

    // make them cry
    cout << john.cry() << endl;
    cout << jack.cry() << endl;

    return 0;
}
```

Copyright © ViSLAB, The University of Sydney

```
   class Trainer {
public:
    void train(Animal animal) {
      cout << animal.cry();
      return;
    }
};

int main() {
    Brother john("John");
    Dog jack("Jack");

    Trainer trainer;

    // make them cry
    trainer.train(john);
    trainer.train(jack);

   return 0;
}
```

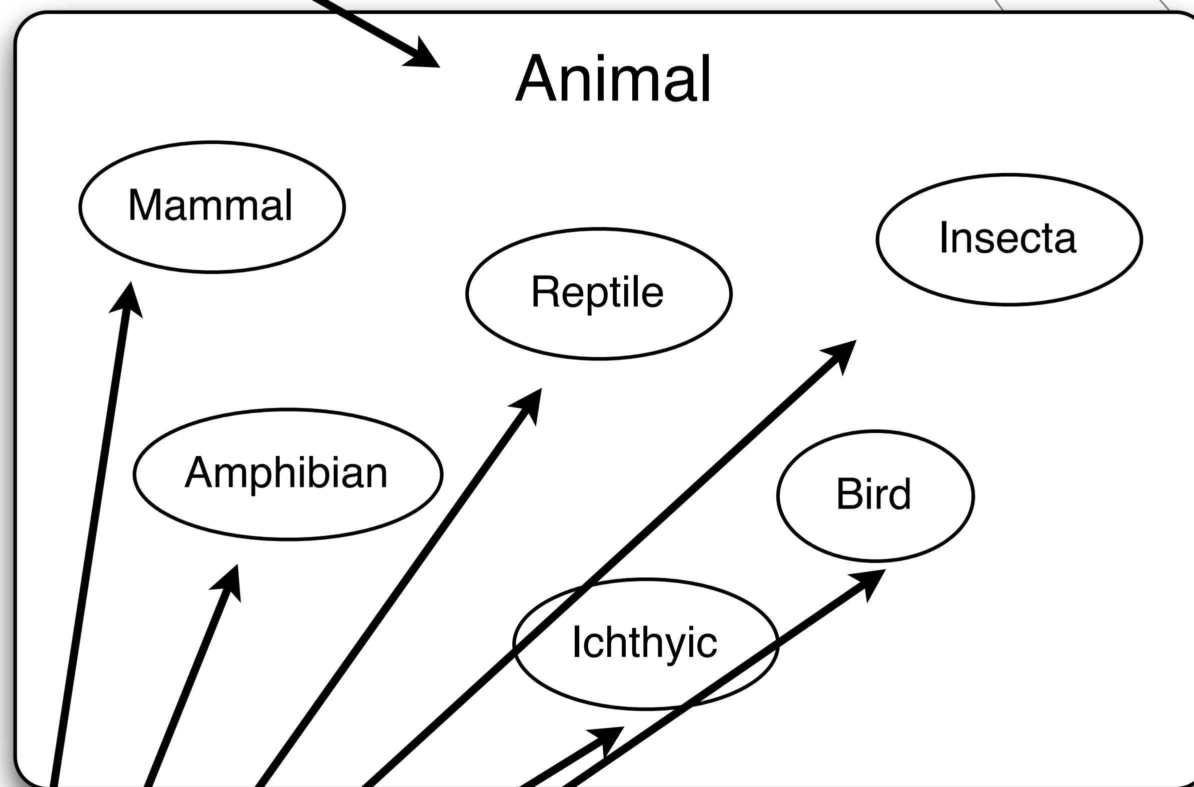
Copyright © ViSLAB, The University of Sydney

Inheritance

- systematically organize common/different features
  - From “Superclass” to “Subclass”
  - Subclass inherits from its Superclass.

Copyright © ViSLAB, The University of Sydney

Superclass



Subclass

Copyright © ViSLAB, The University of Sydney

```
   class Animal {
private:
   string name;
public:
   Animal(string name):name(name) {};
   virtual void move() {};
    virtual string cry() {};
};

class Mammal : public Animal {
   virtual void bear() {};
};

class Bird : public Animal {
   virtual void fly() {};
};
```

- Is anything odd?

Copyright © ViSLAB, The University of Sydney

Don’t be afraid to question.

- Typical OO teaching..
  - Difficult terms...with abstract explanation
  - use of metaphor(s)
  - sample code based on the metaphor
  - but! you still need to support conventional technologies...

Copyright © ViSLAB, The University of Sydney

Class vs. Instance

- Abstract Concept:
  - Class == Kind, Instance == concrete object
- metaphor/example:
  - Animal, Mammal, Bird
- So far, sounds good....

Copyright © ViSLAB, The University of Sydney

```
    #include <string>
#include <iostream>
using namespace std;

class Animal {
private:
    string name;
public:
    Animal(string name):name(name) {}
   virtual string cry() {}
};

// define Brother
class Brother : public Animal {
public:
    Brother(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "*sob*";
    }
};

// define Dog
class Dog : public Animal {
public:
    Dog(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "bowwow";
    }
};

int main() {
    Brother john("John");
    Dog jack("Jack");

    // make them cry
    cout << john.cry() << endl;
    cout << jack.cry() << endl;

     return 0;
}
```

Copyright © ViSLAB, The University of Sydney

Class/Instance which one is first?

- According to the program,
  - Class is first defined, and
  - Instances are created from it.
- If OO is for representing the real world, does this make sense?

Copyright © ViSLAB, The University of Sydney

- In the real world,
  - there are many concrete objecs.
  - depending on how one would see them, one would categolize them based on some criteria.

Copyright © ViSLAB, The University of Sydney

Inconsistency

- Me
  - At Uni : Professor
  - With Clients: Consultant
  - At Home: Dad / Husband
  - At Street: ?

Copyright © ViSLAB, The University of Sydney

Object’s life doesn’t change

- Objects are generated from a Class,
  - Object’s class does not change.
  - example: Baby.
    - Once you create the instance of baby, this baby never become an instance of Adult.
- This problem also apply to “Inheritance”

Copyright © ViSLAB, The University of Sydney

How about Polymorphism?

- A mechanism to provide a common messaging to instances of similar classes ✓
- example of Animal.cry().

Copyright © ViSLAB, The University of Sydney

```
    #include <string>
#include <iostream>
using namespace std;

class Animal {
private:
    string name;
public:
    Animal(string name):name(name) {}
   virtual string cry() {}
};

// define Brother
class Brother : public Animal {
public:
    Brother(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "*sob*";
    }
};

// define Dog
class Dog : public Animal {
public:
    Dog(string n): Animal(n) {}  // constructor (executed at the time of instance creation)
    string cry() {    // make him cry.
        return "bowwow";
    }
};

int main() {
    Brother john("John");
    Dog jack("Jack");

    // make them cry
    cout << john.cry() << endl;
    cout << jack.cry() << endl;

     return 0;
}
```

Copyright © ViSLAB, The University of Sydney

- Unless your brother is big enough to understand you...
- Unless your dog is one of those appear on YouTube as “Amazing Dog”...
  - They would not under stand the message “Cry” in the real world

Copyright © ViSLAB, The University of Sydney

- How about if there were more messages:
  - Wait
  - Eat / Stop eating
  - Clean-up your room/kennel
  - Go toilet, etc.
- Will they do as you say?

Copyright © ViSLAB, The University of Sydney

- In OO world, all behaviours are pre-defined.
- It is understood that all objects faithfully execute the command.

- Polymorphism is based on this basic.
- Hence, message-passing and polymorphism are not for representing the real world as is.

Copyright © ViSLAB, The University of Sydney

- Class, Polymorphism, Inheritance
  - mechanisms for Programming
  - not necessarily for “representing the real world as is”.

Copyright © ViSLAB, The University of Sydney

Simple OO Modelling/Design

- OO : Describe a system with objects and their interaction.
- Analyse the "real objects" and categorize in abstracted manner. (design "Class")
- UML (Unified Modelling Language)
  - became standard in 1997
  - common language

Copyright © ViSLAB, The University of Sydney

You need OOD "training"

- Theory: What is OO, Design Pattern, etc in Textbooks
- Practice: Build a real system
- You need "Training" in between.

Copyright © ViSLAB, The University of Sydney

Design (Modelling (Analysis))

- To do good design,
- you need good modelling
- you need good analysis

Copyright © ViSLAB, The University of Sydney

Training Process

1. Pick an ordinary item/thing around you,
2. Discern the essence of the item/thing
3. Describe it as an abstracted item/thing

- As a training, spend decent amount of time on #2.
- If you reached #3 from #1,
  - list out how you modelled
  - evaluate your model.

Copyright © ViSLAB, The University of Sydney

Example

- Model Ice Cream

images from Google Image search

Copyright © ViSLAB, The University of Sydney

Step 1

- Understand the overall picture.
  - list out everything you can think of..
  - search literature

- understand facts and their relationships with concrete examples

Copyright © ViSLAB, The University of Sydney

Ice cream ?

- kinds:
  - vanilla, choc, mint, etc.
  - waffle cone, cake cone, pretzel cone, kiddie cup.
- structure:
  - ice cream (ball) sitting on the cone
- nature:
  - It melts.

Copyright © ViSLAB, The University of Sydney

Step 2

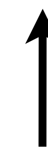
- Decide your view point.
- Decide what are the essence
- Decide how you would represent

- Conceptualisation

Copyright © ViSLAB, The University of Sydney

- Ice cream (ball)
- cone

Ice cream



placed on

cone

Copyright © ViSLAB, The University of Sydney

- Ice cream (ball): different flavours
- cone : different types of cones

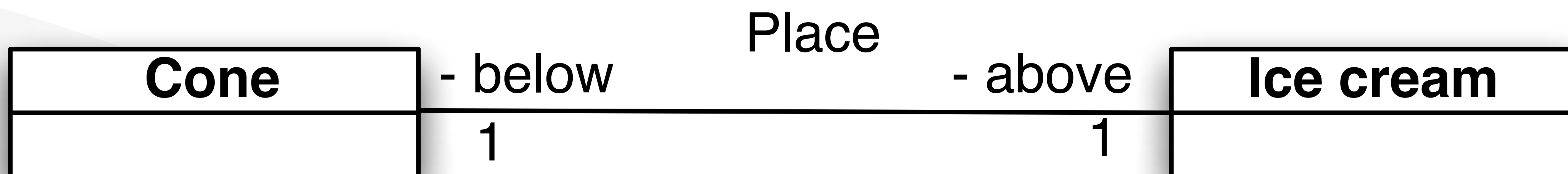
Copyright © ViSLAB, The University of Sydney

- Ice cream cone is
  - different flavours of ice cream ball
  - is placed on
  - different types of cone.

Copyright © ViSLAB, The University of Sydney

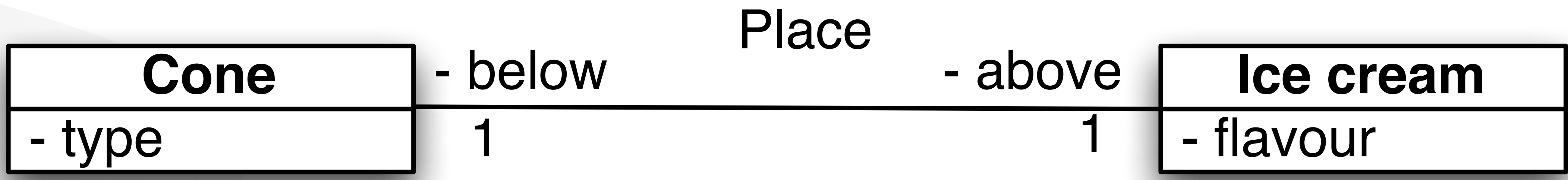
Step 3 (Modelling)

- Represent Structural relationship



Copyright © ViSLAB, The University of Sydney

- Represent each internal features



Copyright © ViSLAB, The University of Sydney

More modelling

- State of Ice cream

frozen

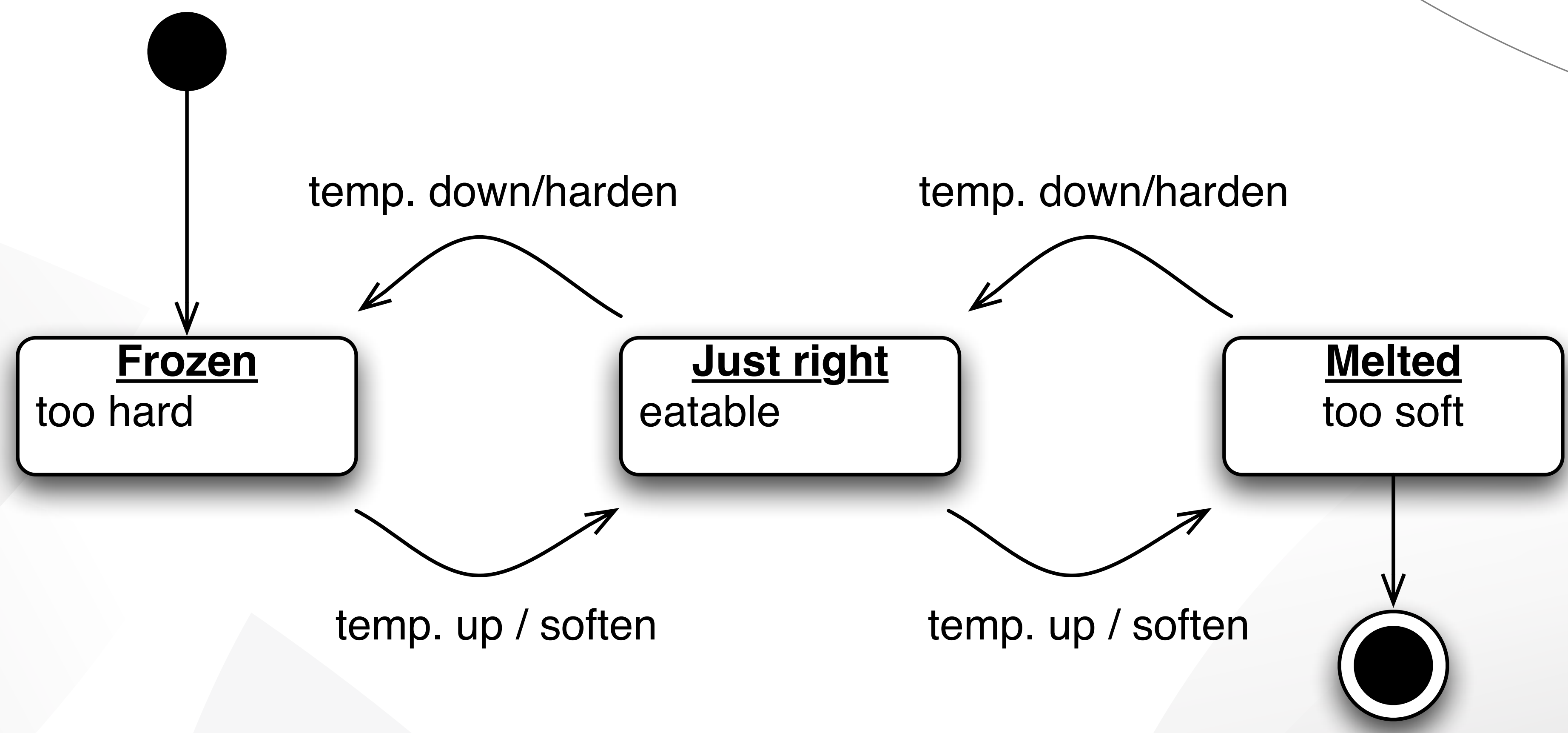
melted

not etable

etable

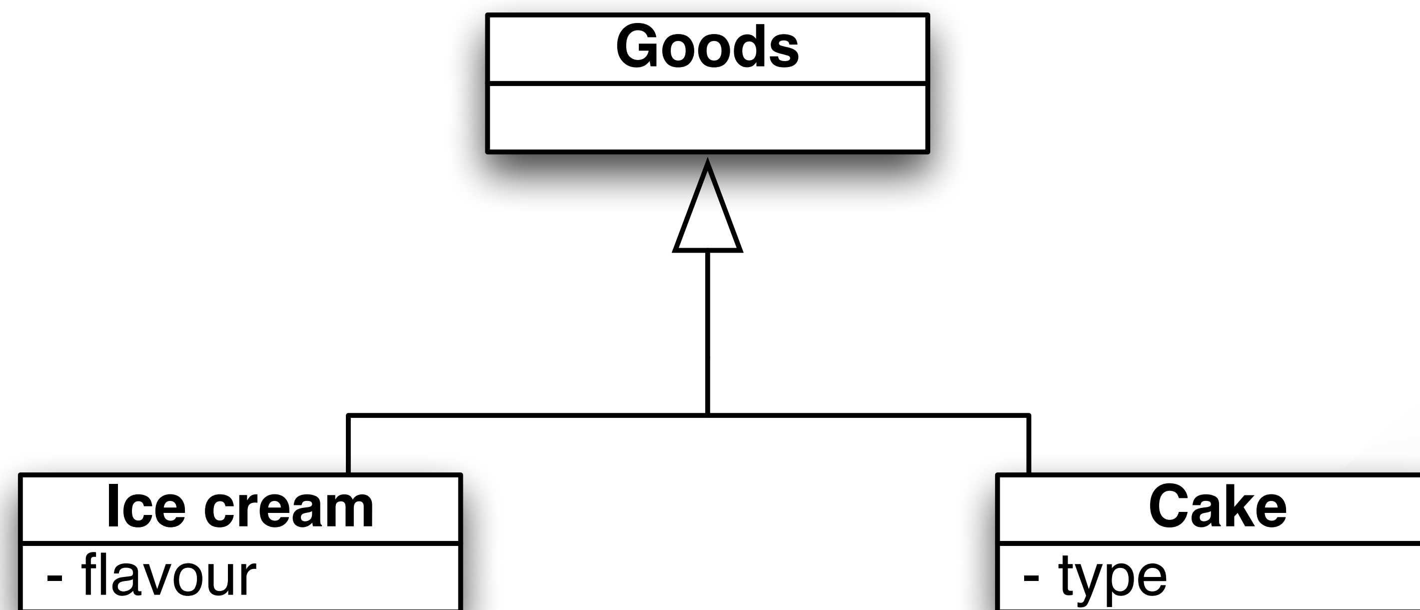
not etable

Copyright © ViSLAB, The University of Sydney



Copyright © ViSLAB, The University of Sydney

- If you change a view point, you would reach a different model.



Copyright © ViSLAB, The University of Sydney