Object Oriented Programming Concepts

Lecture Five: Inheritance – Part One

School of Computing and Information Technology

Faculty of Engineering and Computing

University of Technology, Jamaica

Email: dwwhite@utech.edu.jm, rclarke@utech.edu.jm

Object Oriented Programming Concepts

Expected Outcome

At the end of this lecture the student should be able to:

- Explain how inheritance facilitates software reuse.
- Identify candidates for inheritance relationships during object-oriented analysis.
- Represent inheritance relationships in UML.
- Represent inheritance relationships in code

Object Oriented Programming Concepts

Topics to be covered in this lecture:
Software reuse through inheritance
Base and derived classes
Single and multiple Inheritance
How access specifiers affect inheritance
Identifying inheritance relationships during object-oriented analysis
Inheritance relationships in UML
Inheritance relationships in code

Software reuse through inheritance

Inheritance describes a relationship in which a child class derives members from their parent class(s).

Inheritance is a key concept of OOP.

In Inheritance the state (attributes) and the behaviours (methods) of the parent class are inherited by the child class.

Inheritance can be used to eliminate code redundancy.

And Promotes software reuse.

Software reuse through inheritance cont'd

- Inheritance promotes software reuse through:
  - Where a child class inherits the code already written for their parents.
  - General attributes and methods that apply to all the children are placed in the parent class.
  - Once a parent class has been written and tested, it may be inherited as often as needed by children or grandchildren of the class.
  - Each child class may be modified or specialized without affecting the parent class.

Software reuse through inheritance

- Inheritance, through software reuse:
  - Reduces software development time;
    - Less time is spent writing code.
  - Reduces software development cost.
    - Less code implies less cost
  - Increases software reliability.
  - - If tried, tested and proven parent classes are reused through inheritance, then child classes will inherit the same tried, tested, proven code.

Base and Derived classes

The general members that apply to a set of classes in an inheritance relationship are placed in the parent class called the base.

Hence, the parent is called a generalized class.

Each child class inherits from the parent class and may customize itself to differentiate itself from other child classes, hence it is called a specialization or derived class.

Other synonyms exist for parent and child.

Base and Derived classes

- The terms in the table are used together, so for example, one can speak of parent and child classes, or base and derived classes.

SYNONYMS	**SYNONYMS**
Parent Class	Child Class
Base Class	Derived Class
Super Class	Sub Class
Generalized Class	Specialized Class

Single and Multiple Inheritance

Single inheritance occurs when a child inherits from **only one parent**

Multiple Inheritance occurs when a child inherits from **more than one parent**

Multiple inheritance can cause problems with ambiguity if the same method name is inherited from more than one parent (which one will the child use?)

C++ supports single and multiple inheritance

Java only directly supports single inheritance but ***interfaces*** can simulate multiple inheritance

How access specifiers affect inheritance

All members of a parent class are inherited by a child class, **except** for the constructors and destructor

However, access specifiers can determine if the child class has access to the inherited members

Access specifiers: private, public and protected

How access specifiers affect inheritance

- Private
  - The members of the parent class are inherited but cannot be accessed by the child class
- Public
  - The members of the parent class are inherited and can be accessed by the child class and other classes
- Protected
  - The members of the parent class are inherited and can only be accessed by the child its descendants

Identifying inheritance relationships during object-oriented analysis

When conducting OOA, terms such as “is a” hint at a possible inheritance relationship

For this reason, inheritance is also called an “is-a” relationship or a generalization

For example: A library has books. A text book is a book that covers a particular subject area. A dictionary is a book that contains definitions for a list of words.

Book would be the base class, while dictionary and text book would be child classes of book

Identifying inheritance relationships during object-oriented analysis

- If a set of classes are very similar, containing common attributes and/or methods:
  - Define a parent class
  - Place the common attributes and methods in the parent class
  - Place only the attributes and methods that make each child class unique in their respective class

Identifying inheritance relationships during object-oriented analysis

Example:

A dialog box has a title, x and y coordinates, height, and width, and MessageText. It has ClickClose, ClickMaximize, ClickMinimize and ClickOK operations. A modal form has a title, warning level and MessageText. It has ClickOK and Beep operations.

Perform an OOA on the above to identify the classes, attributes, methods and relationship

Identifying inheritance relationships during object-oriented analysis

- Class: dialog box
- Attributes: x and y coordinates, height, and width.
- Methods: ClickClose, ClickMaximize, ClickMinimize
- Class: modal form
- Attribute: warning level
- Method: Beep
- Class: window
- Attributes: title, MessageText
- Method: ClickOK

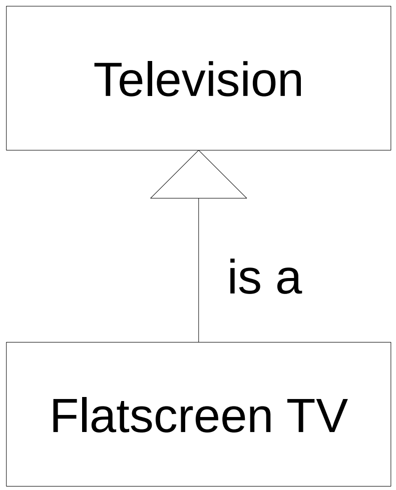
Identifying inheritance relationships during object-oriented analysis

<u>Parent class:</u>	window
<u>Child classes:</u>	modal form, dialog box

- Classes modal form and dialog box inherit the title and Message Text attributes and the ClickOK method from the window class.

Inheritance Relationships in UML

In UML, an inheritance ("is-a" or generalization) relationship is depicted by a solid line connecting the parent and child classes. A hollow triangle (arrow head) is placed on the side of the line connecting the parent class

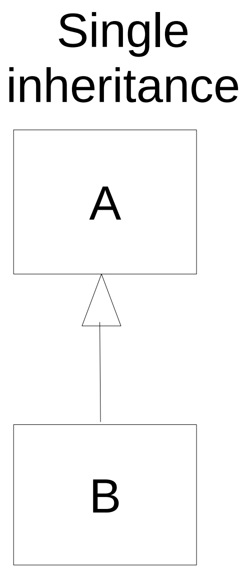


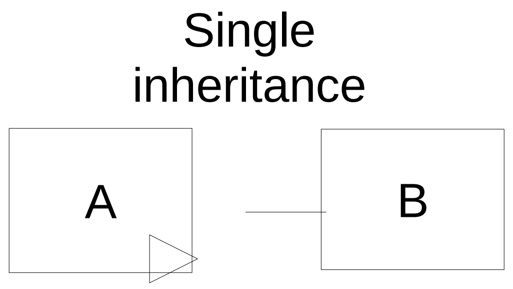
In this example, Flatscreen TV is a Television

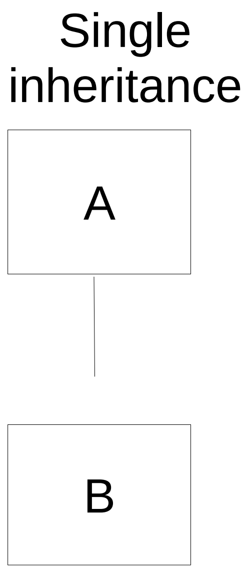
Flatscreen TV is the child class and Television is the parent class

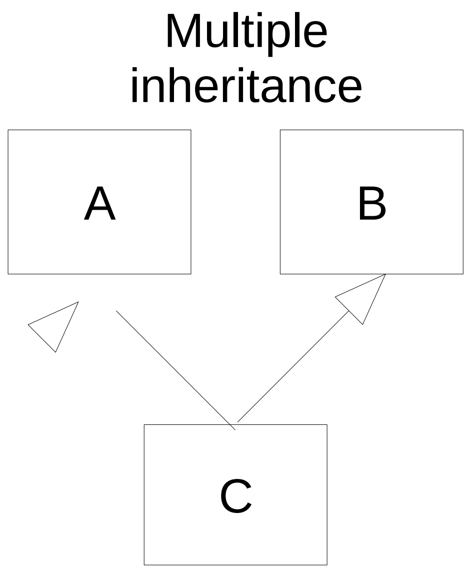
Flatscreen TV inherits the attributes and methods of Television

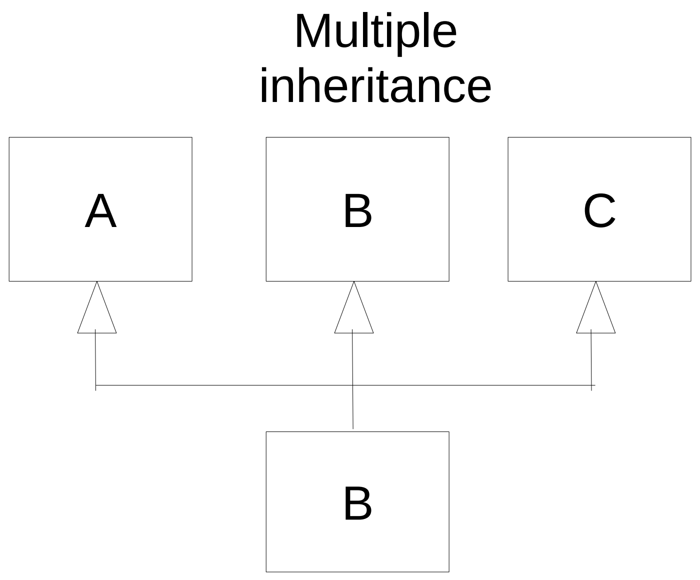
Inheritance Relationships in UML

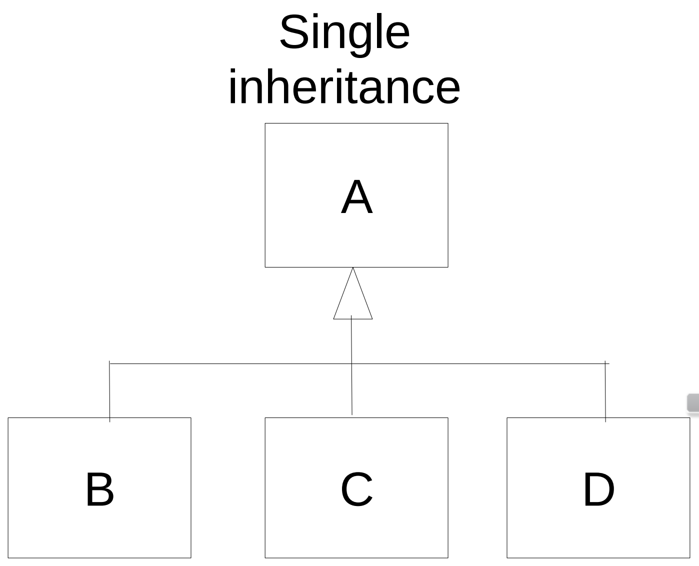






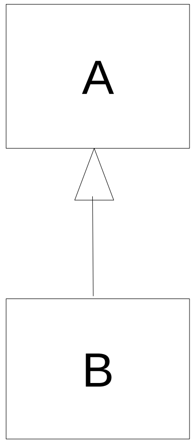




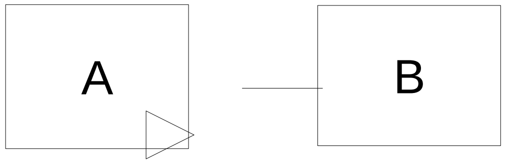


Inheritance Relationships in UML

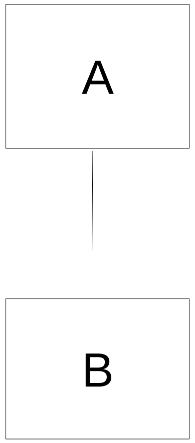
B inherits from A



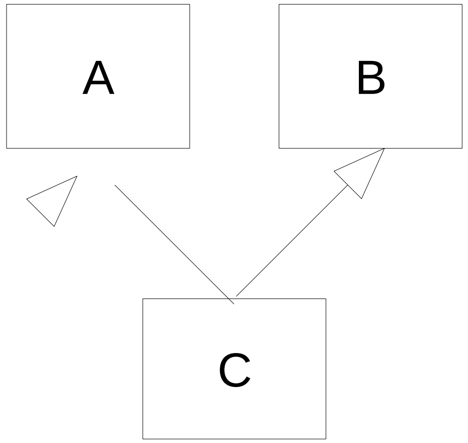
B inherits from A



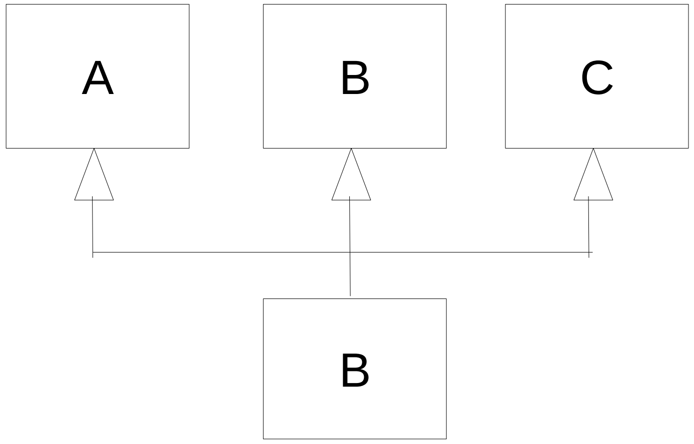
A inherits from B



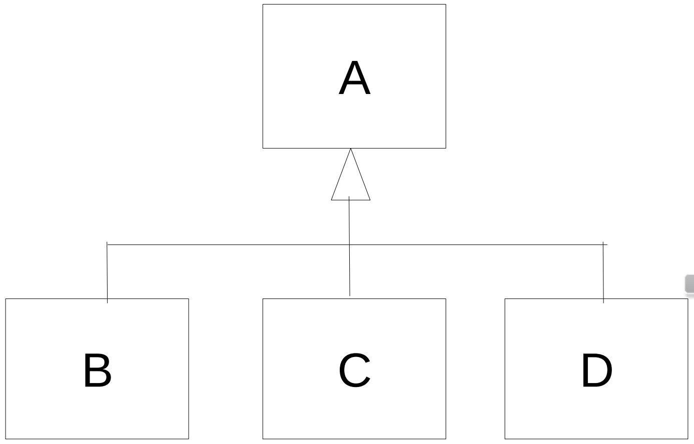
C inherits from A and B



B inherits from A, B and C

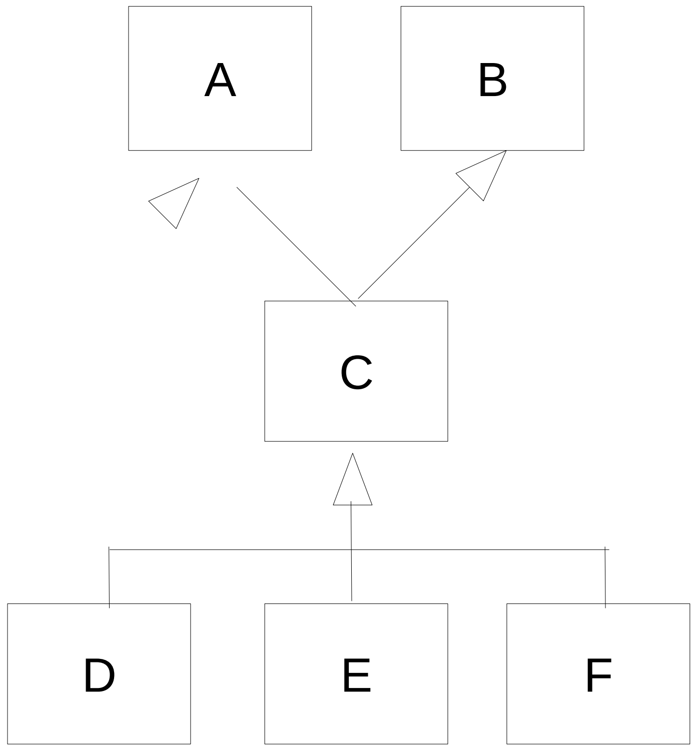


B, C and D inherits from A



Inheritance Relationships in UML

An Inheritance Hierarchy



A and B are both parents and grandparents. They are direct super classes of C, and indirect super classes of D, E and F

C is a child class (sub class) of A and B, but C is also the parent class (super class) of D, E and F

D, E and F are child classes of C which means they are also children of A and B. Therefore D, E and F inherit all the members of A, B and C

Inheritance Relationships in code

C++

If class B inherits from class A:

- Include the header file containing A
  - *#include “A.h”*
- Place a ***colon*** after the name of the child class followed by ***public*** followed by the name of the parent class
  - *class B : public A {*

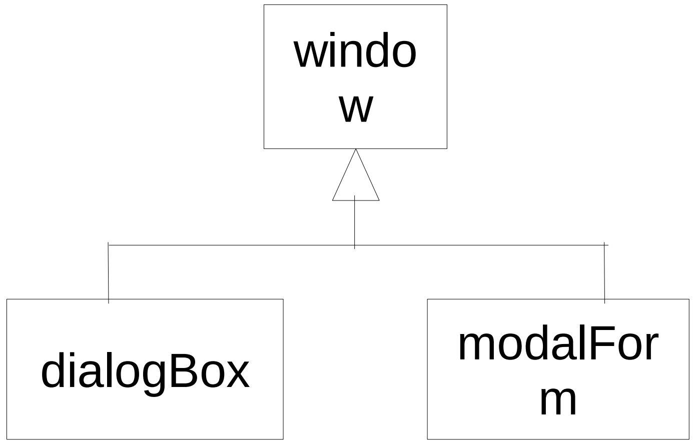
Java

If class B inherits from class A:

- Import the package with the class containing containing A
  - *import PkgA.A;*
- Place the keyword ***extends*** after the name of the child class followed by the name of the parent class
  - *public class B extends A {*

Inheritance Relationships in code

A simple example showing how inheritance is implemented in C++ and Java, based on these UML diagrams



window
#title : string #messageText : string
+ClickOK(): void

modalForm
+warningLevel
+Beep(): void

dialogBox
-x : int -y : int -height : int -width : int
+ClickClose() : void +ClickMaximize() : void +ClickMinimize() : void

The access specifiers in the UML are

•Private	-	(minus sign)
•Public	+	(plus sign)
•Protected	#	(number sign)

A default constructor will be added to the child classes to show how they can access the inherited attributes

Inheritance Relationships in code

C++

```
//window.h
#include <iostream>
#include <string>
using namespace std;

class window {

		protected:
				string title;
				string messageText;

		public:
				void ClickOK()
				{
						cout << "OK was
clicked" << endl;
				}
};
```

Java

```
//window.java
public class window {

		protected String title;
		protected String messageText;

		public void ClickOK()
		{
				System.out.println("OK was
clicked");
		}
}
```

Inheritance Relationships in code

C++

```
//dialogBox.h
#include "window.h"
class dialogBox : public window {
        private:
                int x;
                int y;
                int height;
                int width;
        public:
                void ClickClose()
                {}
                void ClickMaximize()
                {}
                void ClickMinimize()
                {}
                dialogBox()
                {
                        x = 0;
                        y = 0;
                        height = 0;
                        width = 0;
                        title = "";
                        messageText = "";
                }
};
```

Java

```
//dialogBox.java
public class dialogBox extends window  {
        private int x;
        private int y;
        private int height;
        private int width;

        public void ClickClose()
        {}
        public void ClickMaximize()
        {}
        public void ClickMinimize()
        {}
        public dialogBox()
        {
                x = 0;
                y = 0;
                height = 0;
                width = 0;
                title = "";
                messageText = "";
        }
}
```

Inheritance Relationships in code

C++

```
//modalForm.h
class modalForm : public window
{
        private:
                int warningLevel;

        public:
                void Beep()
                {}

                modalForm()
                {
                        warningLevel = 0;
                        title = "";
                        messageText = "";
                }
};
```

Java

```
//modalForm.java
public class modalForm extends window
 {
        private int warningLevel;

        public void Beep()
        {}

        public modalForm()
        {
                warningLevel = 0;
                title = "";
                messageText = "";
        }
}
```

Inheritance Relationships in code

C++

```
//driver.cpp
#include "dialogBox.h"
#include "modalForm.h"

int main ()
{
		dialogBox DB;
		DB.ClickMaximize();
		DB.ClickOK();

		modalForm MF;
		MF.Beep();
		MF.ClickOK();

		return 0;
}
```

Java

```
//driver.java
public class driver
{

	public static void main(String[] args)
	{

		dialogBox DB = new
dialogBox();
		DB.ClickMaximize();
		DB.ClickOK();

		modalForm MF = new
modalForm();
		MF.Beep();
		MF.ClickOK();
	}
}
```