ECE 462
Object-Oriented Programming
using C++ and Java

Program Structure

Yung-Hsiang Lu
yunglu@purdue.edu
Objects and Classes

• Organize objects with similar properties (attributes and behaviors) into classes: human, car, computer, phone, window, circle, rectangle, triangle, bridge, skyscraper ...

• Inheritance: find **commonality** among **classes** (not objects) and create a base class that represents the common **interfaces** and **behavior**

• Common interface of Shape:
  – color
  – line style and thickness
  – area but it is **computed differently in each derived class**
Reuse (Base or Derived?)

• Attribute (member data)
  – If an attribute is shared by all derived classes, the attribute should be declared in the base class. example: color, line style, thickness.
  – If an attribute is unique to a class, it should be declared in this derived class, example: radius for circle.

• Behavior (member function, method):
  – If a method is available to all derived classes, such as getArea and setLineStyle, it should be declared in the base class.
  – If the method’s implementation is applicable to all derived classes, it should be implemented in the base class.
  – If the implementation is unique to each derived class, it should be implemented in the derived classes.
Generate Code from UML Class Diagram
Use Netbeans
• to draw a UML class diagram
• to generate code.
Create a platform-dependent modeling project in the system with empty containers for your model and elements. As you create and populate your model, Java modeling (business) elements are applied to the model.
C++: string (lower case)
Java: String (upper case)

Both languages are case sensitive
attributes

behaviors (functions, operations)

Person

Attributes
private string lastName
private string firstName

Operations
public Person()
public string getLastName()
public void setLastName(string val)
public string getFirstName()
public void setFirstName(string val)
### Person - Properties

<table>
<thead>
<tr>
<th>Operation</th>
<th>Name</th>
<th>Person</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Alias</td>
<td>Person</td>
</tr>
<tr>
<td>Return Type</td>
<td>Return Type</td>
<td>public Person( )</td>
</tr>
<tr>
<td>Parameters</td>
<td>Parameters</td>
<td>public Person( )</td>
</tr>
<tr>
<td>Visibility</td>
<td>Visibility</td>
<td>public</td>
</tr>
<tr>
<td>Documentation</td>
<td>Documentation</td>
<td>public</td>
</tr>
<tr>
<td>Stereotypes</td>
<td>Stereotypes</td>
<td>public</td>
</tr>
<tr>
<td>Tagged Values</td>
<td>Tagged Values</td>
<td>public</td>
</tr>
<tr>
<td>Abstract</td>
<td>Abstract</td>
<td>public</td>
</tr>
<tr>
<td>Final</td>
<td>Final</td>
<td>public</td>
</tr>
<tr>
<td>Static</td>
<td>Static</td>
<td>public</td>
</tr>
<tr>
<td>Native</td>
<td>Native</td>
<td>public</td>
</tr>
<tr>
<td>Strict FP</td>
<td>Strict FP</td>
<td>public</td>
</tr>
<tr>
<td>Concurrency</td>
<td>Concurrency</td>
<td>sequential</td>
</tr>
<tr>
<td>Query</td>
<td>Query</td>
<td>sequential</td>
</tr>
<tr>
<td>Raised Exceptions</td>
<td>Raised Exceptions</td>
<td>sequential</td>
</tr>
<tr>
<td>Constraints</td>
<td>Constraints</td>
<td>sequential</td>
</tr>
<tr>
<td>Redefined</td>
<td>Redefined</td>
<td>sequential</td>
</tr>
</tbody>
</table>

### Parameters

**Parameters**

[Close button]
New Parameter

Name: ln
Type: string

OK  Cancel
Person

Attributes
private string lastName
private string firstName

Operations
public Person(string ln, string fn)
Class Diagram 1

**Person**

**Attributes**
- private string lastName
- private string firstName

**Operations**
- public Person(string ln, string fn)
- public void print()
Generate Java Code from Class Diagram
Person

Attributes
- private String lastName
- private String firstName

Operations
- public Person(String ln, String fn)
- public void print()

Student

Attributes
- private String school
- private String major

Operations
- public Student(String ln, String fn, String sch, String maj)
- public void print()
Open a Java project so that Netbeans can generate code from the UML diagram.
/*
 * To change this template, choose Tools | Templates
 * and open the template in the editor.
 */

package javaapplication15;

/**
 * @author yun-glu
 */

public class Main {

    /**
     * @param args the command line arguments
     */

    public static void main(String[] args) {
        // TODO code application logic here
    }
}

YHL Program Structure 34
package javaapplication15;

public static void main(String[] args) {
    // TODO code application logic here
}

class Main {

    // for yungliu

}
public class Person {

    private String lastName;

    private String firstName;

    public Person(String ln, String fn) {
    }

    public void print () {
    }
}
```java
/*
* To change this template, choose Tools | Templates
* and open the template in the editor.
* /

package javaapplication15;

import Person;
import Student;

/**
* @author yunglu
*/

public class Main {

/**
* @param args the command line arguments
* /

public static void main(String[] args) {
    // TODO code application logic here
    Person p1 = new Person("Johnson", "Tom");
p1.print();

    Student s1 = new Student("Smith", "Mara", "Purdue", "EECS");
}
```
Move class Student

Project: JavaApplication15
Location: Source Packages
To Package: javaapplication15

[Check box] Move Without Refactoring

[Buttons] Preview | Refactor | Cancel | Help
package javaapplication15;

public class Person {

    private String lastName;

    private String firstName;

    public Person(String ln, String fn) {
        lastName = ln;
        firstName = fn;
    }

    public void print() {
        System.out.println("Last name: "+ lastName);
        System.out.println("First name: "+.firstName);
    }
}
public class Student extends Person {

    private String school;

    private String major;

    public Student(String ln, String fn, String sch, String maj) {
        super(ln, fn);
        school = sch;
        major = maj;
    }

    public void print() {
        super.print();
        System.out.println("School: " + school);
        System.out.println("Major: " + major);
    }
}
```java
package javaapplication15;

public class Main {
    public static void main(String[] args) {
        System.out.println("Hello, World!");
    }
}
```
### C++ and Java Syntax

<table>
<thead>
<tr>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>int main(int argc, char * argv[])</td>
<td>public static void main(String[] args) {</td>
</tr>
<tr>
<td>Person p1(&quot;Johnson&quot;, &quot;Tom&quot;);</td>
<td>Person p1 = new Person(&quot;Johnson&quot;, &quot;Tom&quot;);</td>
</tr>
<tr>
<td>p1.print();</td>
<td>p1.print();</td>
</tr>
<tr>
<td>class Person</td>
<td>public class Person {</td>
</tr>
<tr>
<td>{</td>
<td>public Person(String ln, String fn) {</td>
</tr>
<tr>
<td>public:</td>
<td>}</td>
</tr>
<tr>
<td>Person(string ln, string fn);</td>
<td>}</td>
</tr>
<tr>
<td>const string p_lastName;</td>
<td>final String p_lastName;</td>
</tr>
<tr>
<td>class Student: public Person</td>
<td>class Student extends Person</td>
</tr>
</tbody>
</table>
Creating an object must use `new`.

Java

```java
public class FirstMain {

    public static void main(String[] args) {
        Person p1 = new Person("Johnson", "Tom");
        p1.print();
        Student s1 = new Student("Smith", "Mary", "Purdue", "ECE");
        s1.print();
    }

    public class FirstMain {

        public static void main(String[] args) {
            Person p1 = new Person("Johnson", "Tom");
            p1.print();
            Student s1 = new Student("Smith", "Mary", "Purdue", "ECE");
            s1.print();
        }
    }
}
```

main always inside a class

main has only one parameter
main always outside a class

main has two parameters

Creating an object does not use \texttt{new}. (C++ can also use \texttt{new} to create an object; this will be discussed later.)
Encapsulation

• Both C++ and Java provide three levels of protection:
  – private (default), accessible to only the class
  – protected, accessible to the class and its derived classes
  – public, no restriction
• The three protection levels apply to both attributes (also called member data) and functions (also called member function or methods)
• In general, keep as many attributes and functions private as possible. Allow only limited accesses.
Java

package firstjavaproject;

public class Person {
    final String p_lastName;
    final String p_firstName;

    public Person(String ln, String fn) {
        p_lastName = ln;
        p_firstName = fn;
    }

    public void print() {
        System.out.println("Last name: " + p_lastName);
        System.out.println("First name: " + p_firstName);
    }
}

String (uppercase)

public function

no return-type for constructor

no header file for Java

no destructor

no virtual

System.out.println to print
```cpp
#include <string>
using namespace std;

class Person
{
    public:
        Person(string ln, string fn);
        // use at least two letters
        virtual ~Person();
        void print();
    protected:
        const string p_lastName;
        const string p_firstName;
};

// no return-type for constructor
```

- `public:`
- Destructor, no return type, virtual
- String (lowercase)
- No return-type for constructor
```cpp
#include "Person.h"
#include <iostream>

Person::Person(string ln, string fn) :
    p_lastName(ln), p_firstName(fn) {
}

void Person::print() {
    cout << "last name: " << p_lastName << endl;
    cout << "first name: " << p_firstName << endl;
}
```

- Initialization
- Line break
- `cout to print`
Java

extends for derived class

super for base class

constructor

super for base class function
```cpp
#define STUDENT_H
#include "Person.h"
#include <string>
using namespace std;

class Student: public Person {

public:
    Student(string ln, string fn, string sch, string maj):
        virtual ~Student());
    void print(); // same name in Person, override

private:
    string s_school;
    string s_major;

};
#endif /* STUDENT_H */
```

public: for derived class
```cpp
#include "Student.h"
#include <iostream>

Student::Student(string ln, string fn, string sch, string maj):
Person(ln, fn), s_school(sch), s_major(maj) {
}

void Student::print() {
    Person::print();
    cout << "school: " << s_sch << endl;
    cout << "major: " << s_maj << endl;
}
```
What is Method / Message

- A **message** is a request **to an object** to do something. An object can receive a message if it is declared as a method in a base class or the corresponding class.

- In OOP, many objects are created and they interact by “sending messages”, for example,
  - A Driver object sends a message “accelerate” to a MotorVehicle object.
  - An Instructor object sends a message “submitHomework” to a Student object.
  - A Caller object sends a message “callNumber” to a MobilePhone object.
Message

• **not** a network concept
• the mechanism to interact with an object
  – ask a bridge about its length (no parameter)
  – turn on a light (no parameter)
  – accelerate a car (parameter = acceleration, m/s\(^2\))
  – add a customer to a database (parameter = customer)
  – ask a customer of the credit number
• Disallowed messages are checked at **compile time**: compiler error if you ask a light bulb to accelerate.
• A Driver object (dobj) sends an “accelerate” message to a Car object (cobj)
  ⇒ cobj.accelerate();
• A Teacher object (tobj) sends a message to a Student object (sobj) to submit a Homework object (hobj)
  ⇒ sobj.submit(hobj);
• A Person object (pobj) sends a Light Bulb object (bobj) message to turn on
  ⇒ bobj.on();

The object is the **recipient** of the message. Where is the sender? It is implicit by the location of the message.
a Person object can accept "print" message
sent a print message to a Person object
Common Mistakes
A local variable has the same name as an attribute
A local variable has the same name as an attribute

remove final

name not initialized
object declared but not created (in Java)
Self Test
ECE 462
Object-Oriented Programming using C++ and Java

Class Examples

Yung-Hsiang Lu
yunglu@purdue.edu
Overview Package Class Use Tree Deprecated Index Help
PREV CLASS  NEXT CLASS
SUMMARY: NESTED | FIELD | CONSTR | METHOD
FRAMES  NO FRAMES  All Classes
DETAIL: FIELD | CONSTR | METHOD

javax.swing

Class JButton

java.lang.Object
  |-- java.awt.Component
    |-- java.awt.Container
      |-- javax.swing.JComponent
      |-- javax.swing.AbstractButton
          |-- javax.swing.JButton

All Implemented Interfaces:
  ImageObserver, ItemSelectable, MenuComponent

Direct Known Subclasses:
  BasicArrowButton, MetalComboBoxButton

public class JButton
extends AbstractButton
implements Accessible

Done
Overloaded Constructors

overload = several functions with the same name but different parameters

constructor = function of the same name as the class
### Method Summary

<table>
<thead>
<tr>
<th>encapsulation level</th>
<th>return type</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>configurePropertiesFromAction</code></td>
<td>Factory method which sets the AbstractButton's properties according to values from the Action instance.</td>
</tr>
<tr>
<td><code>getUIComponent</code></td>
<td>Returns a string that specifies the name of the L&amp;F class that renders this component.</td>
</tr>
<tr>
<td><code>isDefaultButton</code></td>
<td>Gets the value of the defaultButton property, which if true means that this button is the current default button for its JRootPane.</td>
</tr>
<tr>
<td><code>paramString</code></td>
<td>Returns a string representation of this JButton.</td>
</tr>
<tr>
<td><code>removeNotify</code></td>
<td>Overrides JComponent.removeNotify to check if this button is currently set as the default button on the RootPane, and if so, sets the RootPane's default button to null to ensure the RootPane's ComponentManager is notified.</td>
</tr>
</tbody>
</table>
Resets the UI property to a value from the current look and feel.

Methods inherited from class javax.swing.AbstractButton

addActionListener, addChangeListener, addMouseListener, checkHorizontalKey, checkVerticalKey, createActionListener, createActionPropertyChangeListener, createChangeListener, createItemListener, doClick, doMouseClicked, fireActionPerformed, fireItemStateChanged, fireStateChanged, getAction, getActionCommand, getActionListeners, getChangeListeners, getDisabledIcon, getDisabledSelectedIcon, getDisplayedMnemonicIndex, getHorizontalAlignment, getHorizontalTextPosition, getIcon, getItemListeners, getLabel, getMargin, getMnemonic, getModel, getMultiClickThreshold, getPressedIcon, getRolloverIcon, getRolloverSelectedIcon, getSelectedIcon, getSelectedObjects, getText, getUI, getVerticalAlignment, getVerticalTextPosition, imageUpdate, init, isBorderPainted, isContentAreaFilled, isFocusPainted, isRolloverEnabled, isSelected, paintBorder, removeActionListener, removeChangeListener, removeItemListener, setAction, setActionCommand, setBorderPainted, setContentAreaFilled, setDisabledIcon, setDisabledSelectedIcon, setDisplayedMnemonicIndex, setEnabled, setFocusPainted, setHorizontalAlignment, setHorizontalTextPosition, setIcon, setIconTextGap, setLabel, setLayout, setMargin, setMnemonic, setModel, setMultiClickThreshold, setPressedIcon, setRolloverEnabled, setRolloverIcon, setRolloverSelectedIcon, setSelected, setSelectedIcon,
YPushButton Class Reference

class name, usually noun

The QPushButton widget provides a command button. More...

#include <QPushButton>

Inherits QAbstractButton. inheritance

- List of all members, including inherited members
- Qt 3 support members

Properties

Done

Class Examples
Inherits `QAbstractButton`.

- List of all members, including inherited members
- Qt 3 support members

**Properties**

- `autoDefault` : bool
- `defaultValue` : bool
- `flat` : bool

- 11 properties inherited from `QAbstractButton`
- 56 properties inherited from `QWidget`
- 1 property inherited from `QObject`

**Public Functions**

- `QPushButton` : `QWidget * parent = 0`
- `QPushButton` : `const QString & text, QWidget * parent = 0`
- `QPushButton` : `const QIcon & icon, const QString & text, QWidget * parent = 0`
List of all members, including inherited members
Qt 3 support members

Properties attributes from immediate base class

- autoExclusive : bool
- autoRepeat : bool
- autoRepeatDelay : int
- autoRepeatInterval : int
- checkable : bool
- checked : bool
- down : bool
- icon : QIcon
- iconSize : QSize
- shortcut : QKeySequence
- text : QString

56 properties inherited from QWidget
1 property inherited from QObject

Public Functions

Done
Public Functions

* `QPushButton` (QWidget *`parent = 0`)
* `QPushButton` (const QString &`text`, QWidget *`parent = 0`)
* `QPushButton` (const QIcon &`icon`, const QString &`text`, QWidget *`parent = 0`)
* ~`QPushButton`()
* bool `setDefault` () const
* bool `isDefault` () const
* bool `isFlat` () const
* QMenu *`menu` () const
* void `setAutoDefault` (bool)
* void `setDefault` (bool)
* void `setFlat` (bool)
* void `setMenu` (QMenu *`menu`)

* 21 public functions inherited from QAbstractButton
* 14 public functions inherited from QPushButton
* methods usually verbs
Public Functions

- QWidget (QWidget \* parent = 0, Qt::WindowFlags \*parent 0)
- ~QWidget()
- bool acceptDrops() const
- QString accessibleDescription() const
- QString accessibleName() const
- QList<QAction*> actions() const
- void activateWindow()
- void addAction(QAction \* action)
- void addActions(QList<QAction*> actions)
- void adjustSize()
- bool auxiliaryFillBackground() const
- QPalette::ColorRole backgroundColor() const
- QWidget \* childAt(const QPoint &p) const
- methods usually verbs
- Q
YHL Class Examples 11

- `QRect frameGeometry () const`
- `QSize frameSize () const`
- `const QRect & geometry () const`
- `void getContentsMargins ( int * left, int * top, int * right, int * bottom ) const`
- `virtual HDC getDC () const`
- `void grabKeyboard ()`
- `void grabMouse ()`
- `void grabMouse ( const QCursor & cursor )`
- `int grabShortcut ( const QKeySequence & key, Qt::ShortcutContext context = Qt::WindowShortcut )`
- `bool hasEditFocus () const`
- `bool hasFocus () const`
- `bool hasMouseTracking () const`
- `int heightForWidth ( int w ) const`
- `QWidget methods usually verbs `
- `void insertAction ( QAction * before, QAction * action )`
Naming Convention

• class name: noun, capitalize first letter
• attribute name: noun, lower case
• method name: verb, lower case, followed by upper case
Self Test
ECE 462
Object-Oriented Programming
using C++ and Java

Encapsulation and Polymorphism

Yung-Hsiang Lu
yunglu@purdue.edu
Implementation Independent

• If you ask a student for ID, do you care how the student finds the answer?
  – the student may remember the ID
  – the student may check the student's ID card
  – the student may call the department office and ask
  – the student may call a roommate
  – ...
• For you, these methods are the same. You obtain the ID number of the student.
• You do not need to know how the student implements the method to respond to your question.
Interface vs Implementation

• Suppose you are creating a program to manage books in a library.
  – When the library has only hundreds of books, you can use a list to store each book (author, title, year ...)
  – As the number of books grows, you may need to use a more sophisticated indexing scheme or SQL ....
• The library provides the same interface to search books.
• Users should not feel any difference about the different implementations.
• Encapsulation is enforced by compilers (no run-time surprises).
Interface

• A class' public attributes and methods form the interface for the objects of this class and all derived classes.
• If `func` is a public method, the object must be able to respond to a call of `func`. **This is a "promise".**
• Code reuse \(\Rightarrow\) A promise cannot be withdrawn.
• Anything that is not in the interface (private attributes and methods) can be changed without affecting any code using the class.
\(\Rightarrow\) The program is easier to maintain and reuse.
• Keep an **interface as small as possible.** Make only the essential functionalities visible. Hide all details.
Why Encapsulation?

• prevent accidental modification of objects' attributes
• hide implementation details
• keep data consistency (some attributes must be changed simultaneously)
• reduce the amount of code to maintain (since the details are hidden, they will not affect the users of the code)
• success of OOP
  – large libraries for many functionalities
  – these libraries can be improved without breaking the users' code
Cannot change last name.
The phone number can be changed.
```cpp
#ifndef PERSON_H
#define PERSON_H

#include <string>

using namespace std;

class Person
{

public:
    Person(string ln, string fn);
    // use at least two letters for a variable name
    virtual ~Person();
    void print();

    int p_phoneNumber;

protected:
    string pLastName;
    string psetName;
};

#endif /*PERSON_H__*/
```
add a private attribute
```cpp
#include "Person.h"
#include <iostream>

Person::Person(string ln, string fn, string fa) :
    p_lastName(ln), p_firstName(fn), p_fatherName(fa) {
    
}

Person::~Person() {
}

void Person::print() {
    cout << "last name: " << p_lastName << endl;
    cout << "first name: " << p_firstName << endl;
    cout << "father name: " << p_fatherName << endl;
}
```

change constructor and print
Student::print can access last and first names

```cpp
class Person {
    string ln, fn, fa, sch, maj;
public:
    Person(string ln, string fn, string fa, string sch, string maj); // constructor
    void print(); // function to print student details
};

class Student : public Person {
public:
    Student(string ln, string fn, string fa, string sch, string maj);
    void print() {
        // Person::print();
        cout << "last name: " << p_lastName << endl;
        cout << "first name: " << p_firstName << endl;
        cout << "school: " << s_school << endl;
        cout << "major: " << s_major << endl;
    }
};

int main() {
    Student s("John", "Doe", "Jane", "School", "Computer Science");
    s.print();
    return 0;
}
```

no error
Encapsulation and Polymorphism

```cpp
Student::Student(string ln, string fn, string fa, string sch, string maj) {
    Person(ln, fn, fa), s_school(sch), s_major(maj) {
}

Student::~Student() {
}

void Student::print() {
// Person::print();
    cout << "last name: " << p_lastName << endl;
    cout << "first name: " << p_firstName << endl;
    cout << "father's name: " << p_fatherName << endl;
    cout << "school: " << s_school << endl;
    cout << "major: " << s_major << endl;
}
```

C-Compile [MyFirstCppProject]
Invoking: Cygwin C++ Compiler
g++ -O0 -g3 -Wall -c -fmessage-length=0 -MMD -MP -MF"Student.d" -MT"Student.d" -o"Student.o" ../Student.cpp
../Student.cpp: In member function `void Student::print()':
../Person.h:16: error: `std::string Person::p_fatherName' is private
../Student.cpp:15: error: within this context
make: *** [Student.o] Error 1
Polymorphism
A Student object is always a Person object.
p1.print() behaves as a Student object, even though p1 is declared as a Person object.
s1 = p1 not allowed
A Person object is not necessarily a Student object.
<table>
<thead>
<tr>
<th>Base</th>
<th>Derived</th>
<th>Object</th>
<th>Execute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Y</td>
<td>Y</td>
<td>Base</td>
<td>Base</td>
</tr>
<tr>
<td>Y</td>
<td>Y</td>
<td>Derived</td>
<td>Derived</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>B</td>
<td>B</td>
</tr>
<tr>
<td>Y</td>
<td>N</td>
<td>D</td>
<td>B</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Base</td>
<td>Error</td>
</tr>
<tr>
<td>N</td>
<td>Y</td>
<td>Derived</td>
<td>D</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>B</td>
<td>Error</td>
</tr>
<tr>
<td>N</td>
<td>N</td>
<td>D</td>
<td>Error</td>
</tr>
</tbody>
</table>
Polymorphism

BaseClass obj1 = new BaseClass( ... );
obj1.method();    // call base
DerivedClass obj2 = new DerivedClass( ... );
obj2.method();    // call derived (if available)
obj1 = obj2;      // no problem
obj1.method();    // call derived (if available)
obj2 = obj1;      // error
Virtual Function in C++

- In Java, all functions are "virtual" and polymorphism is always supported.
- In C++, polymorphism is not enabled by default.
- A function is polymorphic only if it is declared virtual at the base class and the object is created by new.
  
  ```cpp
  ClassName * obj = new ClassName(parameters);
  ClassName * obj = new ClassName; // no parameter
  ```

- Once a function is virtual, it is virtual for all derived classes.
- If a derived class can use the same method (implementation), do not override the method.
Virtual in C++

- All virtual methods must have the same prototype (i.e. return type and argument types).
- virtual ⇒ derived class may (not have to) override
- not virtual ⇒ should not override, compiler will allow, but don’t ask for trouble
- why virtual in C++? slightly better performance for non-virtual ... but ... cause too much confusion
- In general, functions in C++ should be virtual unless you have strong reasons (and know what you are doing)
- Constructors are naturally virtual. Destructors should always be virtual (more about this later).
overloaded functions are actually different functions
```cpp
#include "BaseClass.h"
#include <iostream>

using namespace std;
BaseClass::BaseClass() {
}

BaseClass::~BaseClass() {
}

void BaseClass::func1() {
    cout << "BaseClass::func1()" << endl;
}

void BaseClass::func1(int x) {
    cout << "BaseClass::func1(int)" << endl;
}

void BaseClass::func1(double x) {
    cout << "BaseClass::func1(double)" << endl;
}
```
```c++
#ifndef DERIVED1_H
#define DERIVED1_H

#include "BaseClass.h"

class Derived1 : public BaseClass {
public:
    Derived1();
    virtual ~Derived1();
    void func1(int x);
    void func1(double x);
};

#endif /*DERIVED1_H*/
```
#include "Derived1.h"
#include <iostream>

using namespace std;
Derived1::Derived1()
{
}

Derived1::~Derived1()
{
}

void Derived1::func1()
{
    cout << "Derived1::func1()" << endl;
}

void Derived1::func1(int x) {
    cout << "Derived1::func1(int)" << endl;
}

void Derived1::func1(double x) {
    cout << "Derived1::func1(double)" << endl;
}
```cpp
#include "Derived1.h"

class Derived2 : public Derived1 {
public:
    Derived2();
    virtual ~Derived2();
    void func1();
    void func1(int x);
    void func1(double x);
};

#ifdef DERIVED2_H_
#define DERIVED2_H_

#include "Derived1.h"

class Derived2 : public Derived1 {
public:
    Derived2();
    virtual ~Derived2();
    void func1();
    void func1(int x);
    void func1(double x);
};

#endif /*DERIVED2_H_*/
```
```cpp
#include "Derived2.h"
#include <iostream>
using namespace std;

Derived2::Derived2() {}

Derived2::~Derived2() {}

void Derived2::func1() {
    cout << "Derived2::func1()" << endl;
}

void Derived2::func1(int x) {
    cout << "Derived2::func1(int)" << endl;
}

void Derived2::func1(double x) {
    cout << "Derived2::func1(double)" << endl;
}
```
using namespace std;

int main(int argc, char * argv[]) {
    BaseClass * bobj1 = new BaseClass
    // no () since the constructor takes
    bobj1 -> func1();
    bobj1 -> func1(3);
    bobj1 -> func1(5.8);
    cout << endl << endl;
    Derived1 * dobj1 = new Derived1;
    dobj1 -> func1();
    dobj1 -> func1(3);
    dobj1 -> func1(5.8);
    cout << endl << endl;
    bobj1 = dobj1;
    bobj1 -> func1();
    bobj1 -> func1(3);
    bobj1 -> func1(5.8);
    cout << endl << endl;
    Derived2 * dobj2 = new Derived2;
    dobj2 -> func1();
    dobj2 -> func1(3);
    dobj2 -> func1(5.8);
    cout << endl << endl;
    bobj1 = dobj2;
    bobj1 -> func1();
    bobj1 -> func1(3);
    bobj1 -> func1(5.8);
    cout << endl << endl;
}

BaseClass::func1()
BaseClass::func1(int)
BaseClass::func1(double)

Derived1::func1()
Derived1::func1(int)
Derived1::func1(double)

Derived2::func1()
Derived2::func1(int)
Derived2::func1(double)
YHL Encapsulation and Polymorphism

```cpp
 Derived1 * dob1 = new Derived1;
 dob1 -> func1();
 dob1 -> func1(3);
 dob1 -> func1(5.8);
 cout << endl << endl;

 Derived2 * dob2 = new Derived2;
 dob2 -> func1();
 dob2 -> func1(3);
 dob2 -> func1(5.0);
 cout << endl << endl;

 Derived1::func1()
 Derived1::func1(int)
 Derived1::func1(double)
 BaseClass::func1(int)
 BaseClass::func1(double)
 Derived2::func1()
 Derived2::func1(int)
 Derived2::func1(double)
```

Derived2::func1()
Polymorphism is determined at "run-time"
```cpp
#include <iostream>

using namespace std;

class BaseClass {
public:
    void func1(double x) { cout << "BaseClass::func1<double>(double)");
    void func1(int x) { cout << "BaseClass::func1<int>");
    void func1(double x) { cout << "BaseClass::func1<double>");

private:
    double data;
};

class Derived1 : public BaseClass {
public:
    void func1(double x) override { cout << "Derived1::func1<double>");
    void func1(int x) override { cout << "Derived1::func1<int>");
    void func1(double x) override { cout << "Derived1::func1<double>");

private:
    double data1;
};

class Derived2 : public BaseClass {
public:
    void func1(double x) override { cout << "Derived2::func1<double>");
    void func1(int x) override { cout << "Derived2::func1<int>");
    void func1(double x) override { cout << "Derived2::func1<double>");

private:
    double data2;
};

int main(int argc, char *argv[]) {
    Derived2 obj2;
    Derived1 obj1;
    BaseClass obj;

    obj1.func1(5.8);
    obj1.func1(1);
    obj1.func1(3);
    cout << endl << endl;
    obj1 = obj2;

    obj2.func1(5.8);
    obj2.func1(1);
    obj2.func1(3);
    cout << endl << endl;
    obj2 = obj1;

    Derived2 * obj3;
    Derived1 * obj4;
    BaseClass * obj5;

    if (argc > 1) {
        obj3 = new Derived2;
        cout << "obj3 = new Derived2\n";
    } else {
        obj3 = new Derived1;
        cout << "obj3 = new Derived1\n";
    }
    obj3->func1();
    obj3->func1(3);
    obj3->func1(5.8);
    return 0;
}
```

Encapsulation and Polymorphism
Encapsulation and Polymorphism
```cpp
Derived2::func1(ddouble)
Derived2::func1(int)
Derived2::func1(double)

Derived2::func1()
BaseClass::func1(int)
BaseClass::func1(double)

Derived2::func1()
Derived2::func1(int)
Derived1::func1(double)

bobj3 = new BaseClass
BaseClass::func1()
BaseClass::func1(int)
BaseClass::func1(double)
```
```cpp
// main.cpp
#include "Derived2.h"

int main(int argc, char *argv[]) {
    Derived2 obj;
    obj.func1();
    obj.func1(3);
    obj.func1(5.8);
    cout << endl << endl;
    Derived2 d2;
    d2.func1();
    d2.func1(3);
    d2.func1(3);
    d2.func1(5.8);
    cout << endl << endl;
    Derived2 d3;
    d3.func1();
    d3.func1(3);
    d3.func1(3);
    d3.func1(5.8);
    cout << endl;  // Encapsulation and Polymorphism
    return 0;
}
```
no polymorphism if the object is created without using new
Self Test
ECE 462
Object-Oriented Programming using C++ and Java

Constructor and Destructor (C++)

Yung-Hsiang Lu
yunglu@purdue.edu
Constructor (both C++ and Java)

- same name as the class' name (case sensitive)
- no return type
- usually public
- often overloaded
- usually create attribute objects by calling `new`
- "naturally" virtual (C++)
overview Package Class Use Tree Deprecated Index Help

PREV CLASS NEXT CLASS SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAILS: FIELD | CONSTR | METHOD

javax.swing

Class JLabel

java.lang.Object
  | java.awt.Component
  | java.awt.Container
  | javax.swing.JComponent
  | javax.swing.JLabel

All Implemented Interfaces:
  ImageObserver, MenuContainer, Serializable, Accessible, SwingConstants

Direct Known Subclasses:
  BasicComboBoxRenderer, DefaultListCellRenderer, DefaultTableCellRenderer, DefaultTreeCellRenderer

public class JLabel

extends JComponent

implements SwingConstants, Accessible
### Constructor Summary

- **JLabel**
  - Creates a JLabel instance with no image and with an empty string for the title.

- **JLabel**
  - Creates a JLabel instance with the specified image.

- **JLabel**
  - Creates a JLabel instance with the specified image and horizontal alignment.

- **JLabel**
  - Creates a JLabel instance with the specified text.

- **JLabel**
  - Creates a JLabel instance with the specified text, image, and horizontal alignment.

- **JLabel**
  - Creates a JLabel instance with the specified text and horizontal alignment.

### Method Summary

- **protected int checkHorizontalAlignment(int key, String message)**
  - Verify that key is a legal value for the horizontalAlignment properties.
Constructors are usually public.

public JLabel(String text,
              Icon icon,
              int horizontalAlignment)

Creates a JLabel instance with the specified text, image, and horizontal alignment. The label is centered vertically in its display area. The text is on the trailing edge of the image.

Parameters:
  text - The text to be displayed by the label.
  icon - The image to be displayed by the label
  horizontalAlignment - One of the following constants defined in SwingConstants: LEFT, CENTER, RIGHT, LEADING or TRAILING.
QLabel Class Reference

[QtGui module]

The QLabel widget provides a text or image display. More...

```cpp
#include <QLabel>
```

Inherits QFrame.

- List of all members, including inherited members
- Qt 3 support members

**Properties**

- `alignment` : Qt::Alignment
- `indent` : int
- `margin` : int
- `scaledContents` : bool
- `text` : QString
- `textFormat` : Qt::TextFormat
• 6 properties inherited from QFrame
• 56 properties inherited from QWidget
• 1 property inherited from QObject

Public Functions

`QLabel constructor overloaded, public`

• `QLabel` (QWidget *parent = 0, Qt::WindowFlags f = 0)
• `QLabel` (const QString &text, QWidget *parent = 0, Qt::WindowFlags f = 0)
• `~QLabel`
• `Qt::Alignment alignment()` const
• `QWidget *buddy()` const
• `bool hasScaledContents()` const
• `int indent()` const
• `int margin()` const
• `QMovie *movie()` const
• `bool openExternalLinks()` const
• `const QPicture *picture()` const
• `const QPixmap *pixmap()` const
• `void setAlignment(Qt::Alignment)`
• `void setBuddy(QWidget *buddy)`
• `void setIndent(int)`
• `void setMargin(int)`
Object Creation

- Java:
  
  ```java
  ClassName obj = new ClassName(parameters);
  ```

- C++

  ```cpp
  ClassName obj(parameters);
  ```

  or

  ```cpp
  ClassName * obj = new ClassName(parameters);
  ```

  **notice** *

- In C++ and Java, a class is also a type (similar to int, char, or float).
"new" = Memory Allocation

- In Java, unused memory will be automatically reclaimed (called **garbage collection**).

```java
AClass x1 = new AClass(10, 100);
x1 = new AClass(20, 200);
```

- It is unnecessary to worry about the objects that **cannot be reached**. Java reclaims the memory.
- You have to **remove all references** to the object.
- Too much garbage, however, **degrades performance**.
C++ Memory Leak

• In C++, unreachable memory is lost, "memory leak".

    AClass * x1 = new AClass( 10, 100 ); // x1 is a pointer
    x1 = new AClass( 20, 200 );

• To prevent memory leak
  delete x1;

• Do not use malloc / free in C++. They do not call the constructor / destructor.
new - delete

- If an object is created by calling "new", it should be removed by calling "delete".
  
  ```
  AClass * x1 = new AClass( 10, 100 );
  delete x1;
  ```

- new – delete pair
  - call delete only if new is called earlier
  - in the same level (not necessarily the same function)
  - an object can be deleted only once

- use “valgrind --tool=memcheck --leak-check=yes executable” to check memory leak
Memory Leak

• If allocated memory is not reclaimed, the program will gradually run out of memory and eventually crash.
• Memory leak is hard to detect by running the program
  – usually leak a small amount each time
  – can take hours, days, or weeks to run out of memory
  – sometime mistaken as performance problems
• In Java, if an object is no longer needed, remove all references to the object. It will be garbage collected.
• In C++, if an object is no longer needed and the object is created by calling new, delete the object.
Valgrind is an award-winning instrumentation framework for building dynamic analysis tools. There are Valgrind tools that can automatically detect many memory management and threading bugs, and profile your programs in detail. You can also use Valgrind to build new tools.
The Valgrind Quick Start Guide

Release 3.3.0 7 December 2007

Copyright © 2000-2007 Valgrind Developers

Email: valgrind@valgrind.org

Table of Contents

The Valgrind Quick Start Guide
1. Introduction
Use of -O2 and above is not recommended as Memcheck occasionally reports uninitialised-value errors which don't really exist.

3. Running your program under Memcheck

If you normally run your program like this:

```c
myprog arg1 arg2
```

Use this command line:

```c
valgrind --leak-check=yes myprog arg1 arg2
```

Memcheck is the default tool. The `--leak-check` option turns on the detailed memory leak detector.

Your program will run much slower (eg. 20 to 30 times) than normal, and use a lot more memory. Memcheck will issue messages about memory errors and leaks that it detects.

4. Interpreting Memcheck's output

Here's an example C program with a memory error and a memory leak.

```c
#include <stdlib.h>

void f(void)
```
Destructor (C++ only)

- add `~` in front of the class' name
- no return type
- usually public
- **cannot** take parameters ⇒ **cannot** be overloaded
- usually symmetric to constructor
- remove attribute objects by calling `delete` (if they are created by calling `new`)
- should always be `virtual`
Calling Order in Class Hierarchy

- In both C++ and Java
  - the constructor of the base class is called first
  - the constructor of the derived class is called later
  ⇒ Constructors are naturally virtual because the constructor in the derived class is always called.
  ⇒ C++ cannot add virtual in front of a constructor.

- In C++
  - the destructor of the derived class is called first
  - the destructor of the base class is called later
  ⇒ need to add virtual to ensure the derived class' destructor is called
Constant Attributes

- Constant attributes must be initialized in constructors.
- C++
  ```
  const type attributeName;
  constructor(...): attributeName(value) {
      ...
  }
  ```
- Java
  ```
  final type attributeName;
  ⇒ assign value in constructor
  ```
```cpp
#include "BaseClass.h"
#include "Derived1.h"
#include "Derived2.h"
#include <iostream>

using namespace std;

void createObject1() {
    cout << "---A" << endl;
    BaseClass * bobj1 = new BaseClass;
    cout << "---B" << endl;
    BaseClass * bobj2 = new BaseClass(5);
    cout << "---C" << endl;
    Derived1 * dobj1 = new Derived1;
    cout << "---D" << endl;
    Derived1 * dobj2 = new Derived1(7);
    cout << "---E" << endl;
    // objects must be explicitly destroyed
    delete bobj1;
    cout << "---F" << endl;
    delete bobj2;
    cout << "---G" << endl;
    delete dobj1;
    cout << "---H" << endl;
    delete dobj2;
}
```
```cpp
void createObject2() {
    cout << "---01" << endl;
    BaseClass bobj1;
    cout << "---02" << endl;
    BaseClass bobj2(5);
    cout << "---03" << endl;
    Derived1 dobj1;
    cout << "---04" << endl;
    Derived1 dobj2(7);
    // object are automatically destroyed
}

void createObject3() {
    // this will cause memory leak
    cout << "---I" << endl;
    BaseClass * bobj1 = new BaseClass;
    cout << "---II" << endl;
    BaseClass * bobj2 = new BaseClass(5);
}

int main(int argc, char * argv[]) {
    createObject1();
    createObject2();
    createObject3();
    return 0;
}
```
overloaded constructor
```cpp
#include "BaseClass.h"
#include <iostream>

using namespace std;

BaseClass::BaseClass() {
    cout << "BaseClass::BaseClass()" << endl;
}

BaseClass::BaseClass(int x) {
    cout << "BaseClass::BaseClass(int)" << endl;
}

BaseClass::~BaseClass() {
    cout << "BaseClass::~BaseClass()" << endl;
}
```
```cpp
#ifndef DERIVED1_H
#define DERIVED1_H

#include "BaseClass.h"

class Derived1 : public BaseClass {

public:
    Derived1();
    Derived1(int x);
    ~Derived1();
};

#endif /* DERIVED1_H */
```
```cpp
#include "Derived1.h"
#include <iostream>

using namespace std;

Derived1::Derived1() {
    cout << "Derived1::Derived1()" << endl;
}

Derived1::Derived1(int x) {
    cout << "Derived1::Derived1(int)" << endl;
}

Derived1::~Derived1() {
    cout << "Derived1::~Derived1()" << endl;
}
```
#ifndef DERIVED2_H
#define DERIVED2_H

#include "Derived1.h"

class Derived2 : public Derived1 {
public:
    Derived2();
    Derived2(int x);
    virtual ~Derived2();
};

#endif /*DERIVED2_H */
```cpp
#include "Derived2.h"
#include <iostream>

using namespace std;

Derived2::Derived2() {
    cout << "Derived2::Derived2()" << endl;
}

Derived2::Derived2(int x) {
    cout << "Derived2::Derived2(int)" << endl;
}

Derived2::~Derived2() {
    cout << "Derived2::~Derived2()" << endl;
}
```

```cpp
#include "BaseClass.h"
#include "Derived1.h"
#include "Derived2.h"
#include <iostream>
using namespace std;

void createObject1() {
    cout << "---A" << endl;
    BaseClass * bobj1 = new BaseClass;
    cout << "---B" << endl;
    Derived1 * dobj1 = new Derived1;
    cout << "---C" << endl;
    Derived1 * dobj2 = new Derived1(7);
    cout << "---D" << endl;
    // objects must be explicitly destroyed
    delete bobj1;
    cout << "---E" << endl;
}

void createObject2() {
    cout << "---01" << endl;
    BaseClass bobj1;
    cout << "---02" << endl;
    delete dobj2;
}
```

**base constructor first**

**base destructor second**
objects automatically destroyed

destructor not called, memory leak
[(msee190pc9) ~/cpp/ConstructorDestructor/ ] ls
BaseClass.cpp  CVS/ Derived1.h  Derived2.h
BaseClass.h    Derived1.cpp Derived2.cpp  main.cpp
[(msee190pc9) ~/cpp/ConstructorDestructor/ ]
ls
BaseClass.cpp  CVS/  Derived1.h  Derived2.h
BaseClass.h  Derived1.cpp  Derived2.cpp  main.cpp

qmake -project
qmake
make
[(msee190pc9) ~/cpp/ConstructorDestructor/ ] valgrind --leak-check=yes ./ConstructorDestructor
leak detected

loc.c:168)
==15112== by 0x4016c2: createObject3() (in /home/shay/a/ee462b30/cpp/ConstructorDestructor/ConstructorDestructor)
==15112== by 0x401712: main (in /home/shay/a/ee462b30/cpp/ConstructorDestructor/ConstructorDestructor)
==15112==
==15112== 8 bytes in 1 blocks are definitely lost in loss record 2 of 2
==15112== at 0x4904db5: operator new(unsigned long) (vg_replace_malloc)
loc.c:168)
==15112== by 0x401696: createObject3() (in /home/shay/a/ee462b30/cpp/ConstructorDestructor/ConstructorDestructor)
==15112== by 0x401712: main (in /home/shay/a/ee462b30/cpp/ConstructorDestructor/ConstructorDestructor)
==15112==
==15112== LEAK SUMMARY:
==15112== definitely lost: 16 bytes in 2 blocks.
==15112== possibly lost: 0 bytes in 0 blocks.
==15112== still reachable: 0 bytes in 0 blocks.
==15112== suppressed: 0 bytes in 0 blocks.
==15112== Reachable blocks (those to which a pointer was found) are not shown.
==15112== To see them, rerun with: --show-reachable=yes [(msee190pc9) ~/cpp/ConstructorDestructor/]
```cpp
    cout << "---II" << endl;
    BaseClass * obj2 = new BaseClass(5);
}

int main(int argc, char * argv[]) {
    createObject1();
    createObject2();
    // createObject3();
    return 0;
}
```
YHL Constructor and Destructor
---01
BaseClass::BaseClass()
---02
BaseClass::BaseClass(int)
---03
BaseClass::BaseClass()
Derived1::Derived1()
---04
BaseClass::BaseClass()
Derived1::Derived1(int)
Derived1::~Derived1()
BaseClass::BaseClass()
Derived1::~Derived1()
BaseClass::~BaseClass()
BaseClass::~BaseClass()
BaseClass::~BaseClass()
==15200==
==15200== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 5 from 2)
==15200== malloc/free: in use at exit: 0 bytes in 0 blocks.
==15200== malloc/free: 16 allocs, 16 frees, 464 bytes allocated.
==15200== For counts of detected errors, rerun with: -v
==15200== All heap blocks were freed -- no leaks are possible.
[(msee190pc9) ~/cpp/ConstructorDestructor/]
Derived1's constructor called

Derived1's destructor called
`BaseClass` destructor is not virtual.

```cpp
class BaseClass
{
public:
    BaseClass();
    BaseClass(int x);
   // virtual
    ~BaseClass();
};
```
Derived1's constructor called

Derived1's destructor not called

⇒ possible memory leak
Destructor cannot have parameters

Destructor cannot be overloaded
```cpp
#ifndef BASECLASS_H
#define BASECLASS_H

class BaseClass {
public:
    BaseClass();
    BaseClass(int x);
    virtual ~BaseClass();
    // virtual ~BaseClass(int x);
private:
    const int cValue;
};
#endif /*BASECLASS_H*/
```
const attribute must be initialized in constructor
const attribute cannot be initialized using assignments
const attribute must be initialized using ":="
package javaconstructor;

public class JavaMain {

/**
 * @param args
 */

    public static void main(String[] args) {
        // TODO Auto-generated method stub
        System.out.println("A");
        BaseClass bobj1 = new BaseClass();
        System.out.println("B");
        BaseClass bobj2 = new BaseClass(4);
        System.out.println("C");
        BaseClass bobj3 = new Derived1Class();
        System.out.println("D");
        BaseClass bobj4 = new Derived2Class(4);
    }
}
package javaconstructor;

public class BaseClass {
    public BaseClass() {
        System.out.println("BaseClass::BaseClass()");
    }

    public BaseClass(int x) {
        System.out.println("BaseClass::BaseClass(int"));
    }
}
package javaconstructor;

public class Derived1Class extends BaseClass {

    public Derived1Class() {
        System.out.println("Derived1Class::Derived1Class()");
    }

    public Derived1Class(int x) {
        System.out.println("Derived1Class::Derived1Class(int)");
    }
}

no need to call super
package javaconstructor;

public class Derived2Class extends Derived1Class {
    public Derived2Class() {
        System.out.println("Derived2Class::Derived2Class() ");
    }

    public Derived2Class(int x) {
        System.out.println("Derived2Class::Derived2Class(int) ");
    }
}
class JavaMain {
  public static void main(String[] args) {
    System.out.println("A");
    BaseClass bobj2 = new BaseClass(4);
    System.out.println("B");
    BaseClass bobj3 = new Derived1Class();
    System.out.println("C");
    BaseClass bobj4 = new Derived2Class(4);
  }
}

base constructor first
package javaconstructor;

public class Derived1Class extends BaseClass {

    public Derived1Class() {
        System.out.println("Derived1Class::Derived1Class() ");
    }

    public Derived1Class(int x) {
        super(x);
        System.out.println("Derived1Class::Derived1Class(int ");
    }
}
package javaconstructor;

public class Derived2Class extends Derived1Class {
  public Derived2Class() {
    System.out.println("Derived2Class::Derived2Class()");
  }

  public Derived2Class(int x) {
    super(x);
    System.out.println("Derived2Class::Derived2Class(int x)");
  }
}

| call super |
Call constructor(int)
constant (final) must be initialized in constructor
package javaconstructor;

public class BaseClass {

    public BaseClass() {
        fValue = 4;
        System.out.println("BaseClass::BaseClass()");
    }

    public BaseClass(int x) {
        fValue = x;
        System.out.println("BaseClass::BaseClass(int)");
    }

    private final int fValue;
}

no problem
The final field BaseClass.fValue cannot be assigned
Self Test