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

Brief History of C++ and Java

Yung-Hsiang Lu
yunglu@purdue.edu
C++ History

• why to study history?
  – Knowing the past often helps us plan for the future.
  – The design decisions of one programming language help us design better languages.

• Since C++ (1982), many new programming languages have been developed:
  – 1991 Python
  – 1995 Java 1
  – 1995 PHP
  – 1997 OO COBOL
  – ...
History of C++: 1979-1991
by Bjarne Stroustrup

• background
  – 1977 Apple 1 & 2 (1MHz processor, 4-48KB memory, $1300-$2600)
  – 1979 Intel 8088
  – 1980 Seagate (then called Shugart) 5.25-in 5MB disk
  – 1981 IBM PC (4.77MHz, 16-640KB memory)
  – 1983 TCP/IP
• Computers were slow and expensive.
A history of C++: 1979–1991

Full text [Pdf] (3.04 MB)

Source History of Programming Languages
The second ACM SIGPLAN conference on History of programming languages
Cambridge, Massachusetts, United States
Pages: 271 - 297
Year of Publication: 1993
ISBN: 0-89791-570-4
Also published in ...

Author Bjarne Stroustrup AT&T Bell Labs, Murray Hill, NJ
Sponsor SIGPLAN: ACM Special Interest Group on Programming Languages
Publisher ACM New York, NY, USA

Bibliometrics Downloads (6 Weeks): 24, Downloads (12 Months): 141, Citation Count: 8

Additional abstract references cited by index terms review collaborative colleagues
C++

- design goals:
  - Simula's facilities for program organization
  - C's efficiency and flexibility
  - for system programming
- 1979-1983 C with Classes
- 1982-1985 C++
- 1985-1988 C++ 2.0
- 1988- standardization (ISO / ANSI)
- ISO = International Organization for Standardization
- ANSI = American National Standards Institute
Simula

- simulator for a distributed system
- class hierarchy
- capturing type errors by compiler
  - type: int, string, Student, Computer ...
  - type error, for example, a Student object + 3, a Computer object + "hello" ...
- problem of Simula: link time too long
  - run-time type checking
  - variable initialization
  - garbage collection, even for a program without garbage
  ⇒ performance too low
Programming Language Design

- Never attack a problem with wrong tools.
- Support for program organization: class, hierarchy, concurrency, static type checking
- Good tools to compile files separately, to link files written in different languages, and to produce fast programs
- Portable across different machines
- His background in OS and communication affects many design decisions, such as model of protection and exception handling

- **A good language requires a good implementation. Performance matters.**
C with Classes

- new language developed to analyze UNIX kernel: analyze network traffic and modularize kernel
  ⇒ develop an extension of C by adding tools
  ⇒ Some programming languages are developed for specific purposes and then are generalized.
- no primitives to express concurrency, use libraries instead (different from Java with built-in thread supports)
  - built-in support: consistent with language, but may cause unnecessary overhead to the users that do not need this feature
  - libraries: more flexibility but increase the overhead in system administration to ensure version compatibility
- C with Classes to be used anywhere C is used ⇒ efficiency requirements eliminate built-in runtime safety checking
Features in C with Classes

- "philosophy": language designers should not force programmers to use a particular style; instead, designers should provide styles, practices, and tools to help programmers avoid well known traps ⇒ C allows low-level operations and type conversions, so does C with Classes

- features (1980): class, derived class, public / private access, constructor / destructor, call and return, friend class, type checking and conversion of function arguments

- features (1981): inline, default arguments, overloading of assignment operator

- C with Classes was implemented as pre-processor of C ⇒ portable across machines ⇒ common approach for language design today

C with Classes → pre-processor → C → C compiler → executable
Design Decisions in C with Classes

- A class is a type.
- Local variables are allocated at stack, not heap ⇒ no need to call garbage collection.
- Default access control is private.
- Static type checking for function arguments and return values.
- Class declarations and function definitions can be in different files (different from Java). Hence, class declaration can be the "interface" (Java distinguishes interface from class).
- "new" calls constructor (not all valid C programs are valid C++ programs).
- Use-defined types (classes) are treated in the same way as the built-in types.
- Function inlining is used to reduce the overhead of calls ⇒ discourage programmers from declaring data members as public.
// classX.h
class X {
public:
    void foo(int, float);
};

// both a class declaration and // interface

// classX.cpp
#include "classX.h"
void X::foo(int a, float b) {
    ...
}

// define the implementation of a // member function
Garbage Collection in C++

• considered until 1985
• inappropriate for a language (C) already had run-time memory management
• GC would degrade performance unacceptably

• Stroustrup stressed that there was no "grand plan" to develop C++. Hence, the usefulness of the language resided on the ability to attract users in Bell Lab by solving their problems, efficiently.
1982 C++

- C with Classes was a "medium success"
- major features:
  - virtual function
  - function and operator overloading
  - reference
  - constant
- virtual function
  - to adapt to similar but different (common base class) types
  - a large if-then-else or switch-case block is undesirable
  - dilemma: allow adaptability by users without allowing the change of base classes (possibly from the library)
void shape::draw() {
    switch (type) {
    case circle:
        // draw a circle
        break;
    case square:
        // draw a square
        break;
    case triangle:
        // draw a triangle
        break;
    }
};

class Shape {
    virtual void draw() = 0;
};
class Circle: public Shape {
    void draw() ... // draw a circle
};
class Square: public Shape {
    void draw() ... // draw a square
};
1986 C++ 2.0

- multiple inheritance, "the fundamental flaw in these arguments is that they take multiple inheritance far too seriously... it is quite cheap... you don't need it very often but when you do it is essential."
- type-safe linkage
- abstract class
- static member functions
- protected members
- overloading ->

- Exception handling was added later.
Summary

• C++ was developed to solve a specific problem: simulating distributed systems
• It is important to choose a good language as the base and build on top of the base; this can obtain immediate tool support.
• Features do not have to be added at once. Most features are added out of necessity, as the basic functionalities are available.
• Separate compilation and linking is critical for developing large-scale programs.
• Performance is essential. Many design decisions are based on the impact of performance.
Brief History of Java 1995-

• started in 1991 and announced in 1995
• Java started as a technology for entertainment "set-top box" to create a language that can run on small portable systems, not intended for system programming (as C++) ... but cable companies were unwilling to support
• The focus then switched to support Internet for processor (hardware) independent and operating-system independent (to be further discussed later)
• need: execute programs from remote machines through the Internet
⇒ A new language is more likely to succeed to solve a new problem. Solving an old problem is harder because of the existing programs and the infrastructures.
• 1994, a "better browser"
Success

• interactive browser:
  – With Java, users can interact with the browser, beyond browse, scroll, and click.
  – Sun Microsystem, as a primarily hardware company, developed Java to create the demand for high-performance networking equipment and computers.
  – 1995/03/23 San Jose Mercury News headline
  – Security is crucial since malicious code can easily propagate through the Internet (different goals from C++)

• widely used on
  – 4.5B devices
  – 1.5B phones
  – printer, webcam, game, car ...
Java: Sun vs. Microsoft

• 2002, Sun filed a lawsuit against Microsoft for violating the license agreement about Java.
• Java was considered a threat to Microsoft's control of the operating system market.
• Sun accused that Microsoft modified Java in Windows and thus made it incompatible with other platform running Java.
• 2004, the two companies settled.

• (background)
  – 1998 US antitrust against Microsoft, settled on 2001/11/02
  – 2003 European Union issued penalty to Microsoft
  – 2000-2002 Internet bubble burst
  – 2008/06/27 Bill Gate’s last day in Microsoft
<table>
<thead>
<tr>
<th>Feature</th>
<th>C++</th>
<th>Java</th>
</tr>
</thead>
<tbody>
<tr>
<td>organization</td>
<td>AT&amp;T Bell Lab</td>
<td>Sun Microsystem</td>
</tr>
<tr>
<td>target environment</td>
<td>system programming</td>
<td>embedded system</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Internet</td>
</tr>
<tr>
<td>base language</td>
<td>C</td>
<td>N/A</td>
</tr>
<tr>
<td>priority</td>
<td>efficiency</td>
<td>security</td>
</tr>
<tr>
<td>growth force</td>
<td>personal computer</td>
<td>Internet</td>
</tr>
<tr>
<td></td>
<td>(to a lesser extent)</td>
<td></td>
</tr>
<tr>
<td>object-oriented</td>
<td>optional</td>
<td>mandatory</td>
</tr>
<tr>
<td>run-time array index checking</td>
<td>N/A</td>
<td>exception</td>
</tr>
<tr>
<td>memory management</td>
<td>destructor</td>
<td>garbage collection</td>
</tr>
<tr>
<td>global base class</td>
<td>N/A</td>
<td>Object</td>
</tr>
<tr>
<td>multiple inheritance</td>
<td>yes</td>
<td>interface</td>
</tr>
<tr>
<td>Feature</td>
<td>C++</td>
<td>Java</td>
</tr>
<tr>
<td>----------------------------------------------</td>
<td>-------------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>concurrency</td>
<td>external library</td>
<td>built-in, thread</td>
</tr>
<tr>
<td>friend function / class</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>parameter passing</td>
<td>value (primitive types), pointer, reference</td>
<td>value (primitive types), reference</td>
</tr>
<tr>
<td>virtual function</td>
<td>explicit</td>
<td>implicit</td>
</tr>
<tr>
<td>separate interface and implementation</td>
<td>yes (.h and .cpp)</td>
<td>N/A</td>
</tr>
<tr>
<td>exception handling</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>function overloading</td>
<td>yes</td>
<td>yes</td>
</tr>
<tr>
<td>default value of function parameters</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>operator overloading</td>
<td>yes</td>
<td>N/A</td>
</tr>
<tr>
<td>graphics library</td>
<td>external</td>
<td>built-in AWT and SWING</td>
</tr>
</tbody>
</table>
Lessons Learned

• A successful language needs a clearly defined target. Creating a new language to replace an existing one is unlikely to succeed.
• Prioritize the requirements: efficiency for C++ and platform neutral for Java
• Tools (compiler, linker, debugger, runtime environment ...) and libraries (graphics, thread ...) are crucial, probably more important than the "elegance" of a language.
• Performance cannot be ignored. Any new language will be compared with C in terms of performance.
• Keep the non-essential portions of the new language the same as a popular existing language. Do not confuse users.
• Be aware of non-technical forces (such as legal issues)
ECE 462
Object-Oriented Programming using C++ and Java

Java Remote Method Invocation

Yung-Hsiang Lu
yunglu@purdue.edu
Rethink Function Call

**Caller**

```java
... 
pi = computePi(40); 
... 
```

**Callee**

```java
LongFloat computePi(int numDigits) 
{
    ... 
    ....
    ....
}
```

The caller does not care how computePi obtain the result.
How to Implement the Callee?

- compute PI
- use a lookup table
- ask another machine to compute

```java
LongFloat computePi(int numDigits) {
    connect to a server
    send the request to the server
    wait for the response
    return the result to the caller
}
```
Remote Method Invocation
An Overview of RMI Applications

RMI applications often comprise two separate programs, a server and a client. A typical server program creates some remote objects, makes references to these objects accessible, and waits for clients to invoke methods on these objects. A typical client program obtains a remote reference to one or more remote objects on a server and then invokes methods on them. RMI provides the mechanism by which the server and the client communicate and pass information back and forth. Such an application is sometimes referred to as a distributed object application.

Distributed object applications need to do the following:
RMI Architecture

- client-server model
- transmit objects to remote Java virtual machine
Compute Engine Example
Demonstration
ls -R
.
CVS/  RMIClient.policy  RMIServer.policy
Makefile  RMIIInterface/
RMIClient/  RMIServer/

./CVS:
Entries  Repository  Root

./RMIClient:
ClientMain.java  ClientPI.java  CVS/

./RMIClient/CVS:
Entries  Repository  Root

./RMIIInterface:
CVS/  EngineInterface.java  TaskInterface.java

./RMIIInterface/CVS:
Entries  Repository  Root

./RMIServer:
CVS/  ServerEngine.java

./RMIServer/CVS:
compile interface and create a jar file
build server
build client

[java] javac -cp ./RMIInterface.jar RMIclient/ClientMain.java
[java] ls RMIclient/
ClientMain.class ClientPI.class
ClientMain.java ClientPI.java
[java]
[ECE 462 ] rmiregistry &
[1] 10934
ServerEngine bound

start server
execute client and print result
java.rmi

Interface Remote

All Known Subinterfaces:
- ActivationInstantiator, ActivationMonitor, ActivationSystem, Activator, DGC, Registry, RMIConnection, RMIServer

All Known Implementing Classes:

public interface Remote
Source Code
# Name Mapping

<table>
<thead>
<tr>
<th>Sun’s Tutorial</th>
<th>This Example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>package</strong> compute</td>
<td><strong>RMIInterface</strong></td>
</tr>
<tr>
<td><strong>package</strong> client</td>
<td><strong>RMIClient</strong></td>
</tr>
<tr>
<td><strong>package</strong> engine</td>
<td><strong>RMIServer</strong></td>
</tr>
<tr>
<td><strong>interface</strong> Compute</td>
<td><strong>EngineInterface</strong></td>
</tr>
<tr>
<td><strong>interface</strong> Task</td>
<td><strong>TaskInterface</strong></td>
</tr>
<tr>
<td><strong>class</strong> ComputeEngine</td>
<td><strong>Server</strong>Engine</td>
</tr>
<tr>
<td><em>Compute</em></td>
<td><em>EngineInterface</em></td>
</tr>
<tr>
<td><strong>class</strong> ComputePi</td>
<td><strong>Client</strong></td>
</tr>
<tr>
<td><strong>class</strong> Pi</td>
<td><strong>Main</strong></td>
</tr>
<tr>
<td>Pi implements Task &lt;BigDecimal&gt;,</td>
<td><strong>ClientPI</strong></td>
</tr>
<tr>
<td>Serializable</td>
<td></td>
</tr>
</tbody>
</table>
package RMIModule;

import java.rmi.Remote;
import java.rmi.RemoteException;

public interface EngineInterface extends Remote {
    <T> T executeTask(TaskInterface<T> t)
        throws RemoteException;
}
package RMIInterface;

public interface TaskInterface<T> {
    T executeCode();
}

package RMIServer;
import java.rmi.RemoteException;
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.rmi.server.UnicastRemoteObject;
import RMIInterface.EngineInterface;
import RMIInterface.TaskInterface;

public class ServerEngine implements EngineInterface {

    public ServerEngine() {
        super();
    }

    public <T> T executeTask(TaskInterface<T> t) {
        return t.executeCode();
    }

    public static void main(String[] args) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "ECE462 Compute";
            EngineInterface engine = new ServerEngine();

            (Unix) -- ServerEngine.java (Java CVS:1.1.1.1 Abbrev) --Ll--Top--
```java
public <T> T executeTask(TaskInterface<T> t) {
    return t.executeCode();
}

public static void main(String[] args) {
    if (System.getSecurityManager() == null) {
        System.setSecurityManager(new SecurityManager());
    }
    try {
        String name = "ECE462 Compute";
        EngineInterface engine = new ServerEngine();
        EngineInterface stub =
            (EngineInterface)
                UnicastRemoteObject.exportObject(engine, 0);
        Registry registry = LocateRegistry.getRegistry();
        registry.rebind(name, stub);
        System.out.println("ServerEngine bound");
    } catch (Exception e) {
        System.err.println("ServerEngine exception:");
        e.printStackTrace();
    }
}
```
import java.rmi.registry.LocateRegistry;
import java.rmi.registry.Registry;
import java.math.BigDecimal;
import java.net.InetAddress;
import RMIInterface.EngineInterface;

public class ClientMain {
    public static void main(String args[]) {
        if (System.getSecurityManager() == null) {
            System.setSecurityManager(new SecurityManager());
        }
        try {
            String name = "ECE462 Computer";
            Registry registry = LocateRegistry.getRegistry(args[0]);
            EngineInterface comp = (EngineInterface)
                registry.lookup(name);
            ClientPI task = new ClientPI(Integer.parseInt(args[1]));
            BigDecimal pi = comp.executeTask(task);
            System.out.println(pi);
        } catch (Exception e) {
            System.err.println("ClientMain exception:");
            e.printStackTrace();
        }
    }
}
package RMIClient;

import RMIInterface.TaskInterface;
import java.io.Serializable;
import java.math.BigDecimal;
import java.net.InetAddress;

public class ClientPI implements TaskInterface<BigDecimal>, Serializable {

    private static final long serialVersionUID = 227L;
    /** constants used in pi computation */
    private static final BigDecimal FOUR =
            BigDecimal.valueOf(4);
    /** rounding mode to use during pi computation */
    private static final int roundingMode =
            BigDecimal.ROUND_HALF_EVEN;
    /** digits of precision after the decimal point */
    private final int digits;

    /**
     * Construct a task to calculate pi to the specified
     * precision.
     */
    public ClientPI(int d) {
        digits = d;
    }
}
```java
public BigDecimal executeCode() {
    return computePi(digits);
}

/**
 * Compute the value of pi to the specified number of
digits after the decimal point. The value is
computed using Machin's formula:
*
pi/4 = 4*arctan(1/5) - arctan(1/239)
*
and a power series expansion of arctan(x) to
* sufficient precision.
*/
public BigDecimal computePi(int digits) {
    int scale = digits + 5;
    BigDecimal arctan1_5 = arctan(5, scale);
    BigDecimal arctan1_239 = arctan(239, scale);
    BigDecimal pi = arctan1_5.multiply(FOUR).subtract(
        arctan1_239).multiply(FOUR);
    return pi.setScale(digits,
        BigDecimal.ROUND_HALF_UP);
}
```
public static BigDecimal arctan(int inverseX,
        int scale)
{
    BigDecimal result, numer, term;
    BigDecimal invX = BigDecimal.valueOf(inverseX);
    BigDecimal invX2 =
        BigDecimal.valueOf(inverseX * inverseX);
    numer = BigDecimal.ONE.divide(invX,
        scale, roundingMode);
    result = numer;
    int i = 1;
    do {
        numer =
            numer.divide(invX2, scale, roundingMode);
        int denom = 2 * i + 1;
        term =
            numer.divide(BigDecimal.valueOf(denom),
                scale, roundingMode);
        if ((i % 2) != 0) {
            result = result.subtract(term);
        } else {
            result = result.add(term);
        }
        i++;
    } while (term.compareTo(BigDecimal.ZERO) != 0);
Object Serialization

1. Why must classes implement `Serializable` in order to be written to an `ObjectOutputStream`?

The decision to require that classes implement the `java.io.Serializable` interface was not made lightly. The design called for a balance between the needs of developers and the needs of the system to be able to provide a predictable and safe mechanism. The most difficult design constraint to satisfy was the safety and security of classes for the Java programming language.

If classes were to be marked as being `Serializable` the design team worried that a developer, either out of forgetfulness, laziness, or ignorance might not declare a class as being `Serializable` and then make that class useless for RMI or for purposes of persistence. We worried that the requirement would place on a developer the burden of knowing how a class was to be used by others in the future, an essentially unknowable condition. Indeed, our preliminary design, as reflected in the alpha API, concluded that the default case for a class ought to be that the objects in the class be `Serializable`. We changed our design only after considerations of security and correctness convinced us that the default had to be that an object not be serialized.

Security restrictions
INT = RMIInterface
SER = RMIServer
CLI = RMIClient

build:
  javac $(INT)/EngineInterface.java $(INT)/TaskInterface.java
  jar cvf RMIInterface.jar $(INT)/*class
  javac -cp ./RMIInterface.jar $(SER)/ServerEngine.java
  javac -cp ./RMIInterface.jar $(CLI)/ClientMain.java $(CLI)/ClientPI.java

clean:
  rm -f RMIInterface.jar
  rm -f $(INT)/*.class
  rm -f $(SER)/*.class
  rm -f $(CLI)/*.class

start:
  rmiRegistry &
  java -cp /home/shay/a/ee462b30/lecturecode/1112/rmi:/home/shay/a/ee462b30/1112/rmi/RMIInterface.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/1112/rmi/RMIInterface.jar -Djava.rmi.server.hostname=qstruct05.ecn.purdue.edu -Djava.security.policy RMIServer.policy RMIServer.ServerEngine

test:
  java -cp /home/shay/a/ee462b30/lecturecode/1112/rmi:/home/shay/a/ee462b30/1112/rmi/RMIInterface.jar -Djava.rmi.server.codebase=file:/home/shay/a/ee462b30/1112/rmi/RMIInterface.jar -Djava.security.policy=RMIClient.policy RMIClient.ClientMain 65
grant codeBase "file:/home/shay/a/ee462b30/lecturecode/1112/rmi" {
    permission java.security.AllPermission;
};
ECE 462
Object-Oriented Programming using C++ and Java

Ray Tracing

Yung-Hsiang Lu
yunglu@purdue.edu
Lesson 1: How Does It Work?

Written by Administrator
Thursday, 10 July 2008 21:01

Article Index
Lesson 1: How Does It Work?
The Ray-Tracing Algorithm in a Nutshell
Implementing Ray-Tracing
Adding Reflection and Refraction
Source Code
All Pages
Ray Tracing
Advantages

- generate high quality images:
  - model different light sources
  - surface properties
Books

- Ray Tracing from the Ground Up by Kevin Suffern
- An Introduction to Ray Tracing
- Object-Oriented Ray Tracing in C++ by Nicholas Wilt
- Realistic Ray Tracing, Second Edition by Peter Shirley and W. Keith Newell
- Practical Ray Tracing in C by Craig A. Lindley
Reverse Ray Tracing
Refraction and Ray Tracing

\[ \frac{\sin \theta_1}{\sin \theta_2} = \frac{\lambda_1}{\lambda_2} \]