Object oriented programming

Java & C++

http://www.iut-orsay.fr/~fournier/Cork/OOP.pdf

Summary

- Some historical points
- Comparison of non-object oriented features
  - simple programming
  - exceptions
  - parameters
- Comparison of Object oriented features
  - inheritance
  - generic programming

History of these languages

- B (Bell labs) (1970) [Ken Thompson]
  - NB (new B) (B + types) (1971) [Dennis Ritchie]
    - C (1972)
      - C++ (1983-1985)[Bjarne Stroustrup]
- Java
  - The green team (1991)
    - 13 people were chartered by Sun to anticipate the "next wave" of computing
    - WebRunner (the HotJava browser), a Java based clone of Mosaic
  - version 1.0a2 (march 1995) => version 1.5…

Basic differences

- C++ is a compiled language
  - running should be fast…
  - supposes expert programmers
  - C++ allows construction of executable stand alone applications or DLL or CGIs
  - no portability…

Basic differences

- Java is an interpreted language
  - running is slow,
  - many controls may be delayed until running,
  - Java allows construction of portable applications, applets, midlets or servlets that always need a host (JRE, browser, server application launcher…)

Common points

- declaring simple objects
  - `int x; float y; double z; short t; long u; char w;`
- but C++ allows unsigned types
  - `unsigned int; unsigned long;`
- Java has the byte type
  - `byte`
- Both know simple arrays using []
  - `int [], float [], …`

Common points

- basic instructions
  - affectation =
  - if (some boolean condition){actions}
  - operators +, -, *, /, ==, !=, <=, <, >=, >, &, |, !, &&, ||, ++, --, <<, >>
  - combinations +=, *=, /=, -=, <<=, >>=
  - while (some boolean condition) {actions}
  - do {actions} while(some boolean condition)
  - for (;;){actions}

Sample 1

- Here is the source of a function :

```
static float average(int notes [], int
  nbNotes){
  float sum=0;
  int i;
  for (i=0; i<nbNotes; i++)
        sum += (float) notes[i];
  return sum/(float)--i;
  }
```

- C++ or Java ?

Example 1 (C++)

```
#include<iostream>
#include<stdlib.h>

static float average(int notes [], int nbNotes){
   float sum=0;
   int i;
   for (i=0; i<nbNotes; i++)
        sum += (float) notes[i];
   return sum/(float)--i;
   }

void main(int argc, char ** argv){
   int notes[argc-1];
   for (int i=0; i<argc; i++)
        notes[i] = atoi(argv[i+1]);
   cout << "average of these " << argc-1 << " notes is : "<<
   average(notes, argc-1) << endl;
   }
```

Example 1 (Java)

```
class SampleC1{

static float average(int notes [], int nbNotes){
   float sum=0;
   int i;
   for (i=0; i<nbNotes; i++)
        sum += (float) notes[i];
   return sum/(float)--i;
   }

public static void main(String [] argv){
   int [] notes = new int[argv.length];
   for (int i=0; i<argv.length; i++)
        notes[i] = Integer.parseInt(argv[i]);
   System.out.println("average of these "+ argv.length+" notes is
   : "+average(notes, argv.length));
   }
}
```

Better Example 1 (Java)

```
class SampleC1Better{

static float average(int notes []){
   float sum=0;
   int i;
   for (i=0; i<notes.length; i++)
        sum += (float) notes[i];
   return sum/(float)--i;
   }

public static void main(String [] argv){
   int [] notes = new int[argv.length];
   for (int i=0; i<argv.length; i++)
        notes[i] = Integer.parseInt(argv[i]);
   System.out.println("average of these "+ argv.length+" notes is
   : "+average(notes));
   }
}
```

What if errors occurred ?

```
void main(int argc, char ** argv){
   int notes[argc-1];
   for (int i=0; i<argc; i++)
       notes[i] = atoi(argv[i+1]);
   cout << "The given notes are : " << endl;
   for (int i=0; i<argc-1; i++)
       cout << notes[i] << " ";
   cout << "average of these " << argc-1 << " notes
   is : "<< average(notes, argc-1) << endl;
   }
```

- command line : sampleC1 a 10 12 8 zzz
- The given notes are :
- 0 10 12 8 0 average of these 5 notes is : 7.5
- **because `atoi` returns 0 if it is unable to convert…**

Errors and Java…

```
java SampleC1Better 10 8 12 aaa
Exception in thread "main"
   java.lang.NumberFormatException: aaa
         at
   java.lang.Integer.parseInt(Integer.java:429)
         at
   java.lang.Integer.parseInt(Integer.java:479)
         at SampleC1Better.main(sampleC1Better.java:14)
```

A reliable solution in Java…

```
public static void main(String [] argv){
   int [] notes = new int[argv.length];
   int correctNotes = 0;
   for (int i=0; i<argv.length; i++)
        try{
        notes[correctNotes] = Integer.parseInt(argv[i]);
        correctNotes++;
   //done only if conversion is successful
        } catch(NumberFormatException ex){}
   System.out.println("average of these "+correctNotes+" notes is
   : "+average(notes));
   }
```

- java SampleC1MuchBetter 10 8 12 aaa
- average of these 3 notes is : 10.0
- is it really correct ?

A **reliable** solution in C++

```
bool isNumber(char * aString){
    for (int i=0; aString[i]; i++)
          if (!isdigit(aString[i]))     return false;
    return true;
    }

void main(int argc, char ** argv){
    int notes[argc-1];
    int correctNotes = 0;
    for (int i=0; i<argc; i++){
          notes[correctNotes] = atoi(argv[i+1]);
          if (isNumber(argv[i+1]))      correctNotes++;
          }
    cout << "The given notes are : " << endl;
    for (int i=0; i<correctNotes-1; i++)
          cout << notes[i] << " ";
    cout << "average of these " << correctNotes-1 << " notes is : "<< average(notes,
    correctNotes-1) << endl;
    }
```

- sampleC1 a 10 12 8 zzz
- The given notes are :
- 10 12 8 average of these 3 notes is : 15

First partial conclusion

- if you use old C functions inside C++ programs, you must take care about exceptional events alone…
- Java is able to take into account some exceptional situations, but not all of them (use of uninitialized elements for example)

C++ may be closer to Java…

```
int isNumber(char * aString){
    for (int i=0; aString[i]; i++)
         if (!isdigit(aString[i])) throw "NumberFormatException";
    return atoi(aString);
    }

void main(int argc, char ** argv){
    int notes[argc-1];
    int correctNotes = 0;
    for (int i=0; i<argc; i++)
         try{
         notes[correctNotes] = isNumber(argv[i+1]);
         correctNotes++;
         }catch(const char * msg){}
    cout << "The given notes are : " << endl;
    for (int i=0; i<correctNotes-1; i++)
         cout << notes[i] << " ";
    cout << "average of these " << correctNotes-1 << " notes is : "<<
    average(notes, correctNotes-1) << endl;
    }
```

Common points

- handling exceptions
  - try{ some actions where exceptions may happen}
  - catch(parameter){actions}
  - catch(parameter){actions}
  - …
  - words are identical : try, catch, throw…

Differences about exceptions

- C++ exceptions may be simple objects (int, char *,…) while Java exceptions are only instances of classes (java.lang.Throwable or derived)
- C++ subroutines may throw exceptions without warning

```
int isNumber(char * aString){
   for (int i=0; aString[i]; i++)
       if (!isdigit(aString[i]))
              throw "NumberFormatException";
   return atoi(aString);
   }
```

Differences about exceptions

- C++ subroutines may announce their ability to throw exceptions :

```
int isNumber(char * aString) throw(const char*){
  for (int i=0; aString[i]; i++)
   if (!isdigit(aString[i]))
         throw "NumberFormatException";
  return atoi(aString);
  }
```

- but it's more a restriction than an announcement…

throw()

```
int isNumber(char * aString)throw(){
   for (int i=0; aString[i]; i++)
       if (!isdigit(aString[i]))
              throw "NumberFormatException";
   return atoi(aString);
   }
sampleC1WE a 10 12 8 zzz
Abort!
Exiting due to signal SIGABRT
Raised at eip=00013576
eax=007efdfc ebx=00000120 ecx=00000000 edx=0000f940
   esi=007f08b8 edi=00000014
```

- we get the same result with throw(int) or throw(char *)

Differences about catching

- two different absolute weapons :
  - C++ : catch(…){}

```
try{
notes[correctNotes] = isNumber(argv[i+1]);
correctNotes++;
}catch(...){}
```

  - Java : catch(Throwable ex){}
    - because all Java exception classes extend the java.lang.Throwable class.

About parameters

```
#include <iostream>
void routine(int someArray[], short someValue){
    cout << endl << "at the beginning of the routine : "<< endl;
    for (int i=0; i<someValue; i++)
          cout << someArray[i] << ", ";
    someArray = new int[10];
    for (int i=0; i<10; i++)  someArray[i]=2*i;
    someValue = 10;
    cout << endl << "before returning of the routine : "<< endl;
    for (int i=0; i<someValue; i++)
          cout << someArray[i] << ", ";
    }
void main(){
    int anArray [20];
    for (int i=0; i<20; i++) anArray[i]=3*i;
    routine(anArray, 20);
    routine(anArray, 20);
    }
at the beginning of the routine :
0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
before returning of the routine :
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
at the beginning of the routine :
0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48, 51, 54, 57,
before returning of the routine :
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
```

About parameters

```
class Example2{
static void routine(int [] someArray, short someValue){
     System.out.println("at the beginning of the routine : ");
     for (int i=0; i<someValue; i++) System.out.print(someArray[i]+" ");
     System.out.println();
     someArray = new int[10];
     for (int i=0; i<10; i++)        someArray[i]=2*i;
     someValue = 10;
     System.out.println("before returning of the routine : ");
     for (int i=0; i<someValue; i++) System.out.print(someArray[i]+" ");
     System.out.println();
     }
public static void main(String [] args){
     int [] anArray = new int[20];
     for (int i=0; i<20; i++) anArray[i]=3*i;
     routine(anArray, (short)20);
     routine(anArray, (short)20);
     }
}
```

```
at the beginning of the routine :
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57
before returning of the routine :
0 2 4 6 8 10 12 14 16 18
at the beginning of the routine :
0 3 6 9 12 15 18 21 24 27 30 33 36 39 42 45 48 51 54 57
before returning of the routine :
0 2 4 6 8 10 12 14 16 18
```

C++ : transmitting parameters

```
#include <iostream>
void routine(int someArray[], int & someValue){
    cout << endl << "at the beginning of the routine : "<< endl;
    for (int i=0; i<someValue; i++)
           cout << someArray[i] << ", ";
    someArray = new int[10];
    for (int i=0; i<10; i++)        someArray[i]=2*i;
    someValue = 10;
    cout << endl << "before returning of the routine : "<< endl;
    for (int i=0; i<someValue; i++)
           cout << someArray[i] << ", ";
    }
    void main(){
    int SIZE = 20;
    int anArray [SIZE];
    for (int i=0; i<SIZE; i++) anArray[i]=3*i;
    routine(anArray, SIZE);
    routine(anArray, SIZE);
    }
```

```
at the beginning of the routine :
0, 3, 6, 9, 12, 15, 18, 21, 24, 27, 30, 33, 36, 39, 42, 45, 48,
   51, 54, 57,
before returning of the routine :
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
at the beginning of the routine :
0, 3, 6, 9, 12, 15, 18, 21, 24, 27,
before returning of the routine :
0, 2, 4, 6, 8, 10, 12, 14, 16, 18,
```

Protecting parameters

- C++ has the "const" attribute which forbids alteration of objects

```
static float average(const int notes [], int nbNotes){
    float sum=0;
    int i;
    for (i=0; i<nbNotes; i++)
          sum += (float) notes[i];
  notes[0]=0;
    return sum/(float)--i;
    }
```

  - it gives the programmer a guaranty he will be warned if he tries to modify something he should not

```
gpp -c sampleC1.C
sampleC1.C: In function `float average(const int
  *, int)':
sampleC1.C:10: assignment of read-only location
```

Common points : declaring base classes

```
#include <iostream>

class Vehicle{
protected :
    float speed;
public :
    void setSpeed(float newSpeed)
          {speed = newSpeed;}
    float getSpeed(){return speed;}
    };

void main(){
    Vehicle myVehicle;
    myVehicle.setSpeed(100.00);
    cout << "the speed of my vehicle
    is " << myVehicle.getSpeed() <<
    endl;
    }
```

```
class Vehicle{
protected float speed;
public void setSpeed(float
    newSpeed)
          {speed = newSpeed;}
public float getSpeed(){return
    speed;}

public static void main(String []
    args){
    Vehicle myVehicle = new
    Vehicle();
    myVehicle.setSpeed((float)100.
    0);
    System.out.println("the speed
    of my vehicle is
    "+myVehicle.getSpeed());
    }
}
```

Differences about declarations (2)

- C++ knows simple objects, instances of classes, pointers on objects, objects' references

```
Vehicle myVehicle;
Vehicle * anAddress = new Vehicle;
Vehicle * anAddress = new Vehicle();
Vehicle & myVehicle = *new Vehicle;
```

- Java only knows simple objects and hidden pointers shown and used like instances

```
Vehicle myVehicle = new Vehicle();
```

Common points about derivation

- in both languages, base classes may be used to hold common parts of objects…
  - Example : cars have an engine, wheels, a number plate, they have the capability to take passengers, to travel from one place to the other…
  - lorries also have an engine, wheels, a number plate, they have the capability to carry heavy objects, to travel from one place to the other …

Bad solution to represent cars and lorries…

```
import java.util.Vector;

class Car{
private int enginePower;
private int numberOfWheels;
private String numberPlate;
private Vector passengerNames;

public void setPower(int somePower){enginePower=somePower;}
public int getPower(){return enginePower;}

public void setWheelsNumber(int number){numberOfWheels=number;}
public int getWheelsNumber(){return numberOfWheels;}

public void setNumberPlate(String aString){numberPlate=aString;}
public String getNumberPlate(){return numberPlate;}
```

Bad solution (2)

```
public void addPassenger(String hisName){
   if (passengerNames == null) passengerNames=new Vector();
   passengerNames.add(hisName);
   }

public void removePassenger(int hisNumber)
   throws ArrayIndexOutOfBoundsException{
        if (passengerNames==null)
                throw new ArrayIndexOutOfBoundsException();
        passengerNames.remove(hisNumber);
   }

public int getNumberOfPassengers(){
   if (passengerNames==null) return 0;
   return passengerNames.size();
   }
}
```

Bad solution (3)

```
import java.util.Vector;

class Lorry{
private int enginePower;
private int numberOfWheels;
private String numberPlate;
private int carryingCapacity, availableCapacity;

public void setPower(int somePower){enginePower=somePower;}
public int getPower(){return enginePower;}

public void setWheelsNumber(int number){numberOfWheels=number;}
public int getWheelsNumber(){return numberOfWheels;}

public void setNumberPlate(String aString){numberPlate=aString;}
public String getNumberPlate(){return numberPlate;}
```

Bad solution (4)

```
public void loadObject(int itsVolume) throws Exception{
   if (itsVolume <= 0) throw new Exception("BUG");
   if (itsVolume > availableCapacity)
        throw new Exception("Object can't be loaded");
   availableCapacity -= itsVolume;
   }

public void unloadObject(int itsVolume) throws Exception{
   if (itsVolume <= 0 ||
        itsVolume > carryingCapacity-availableCapacity)
              throw new Exception("BUG");
   availableCapacity += itsVolume;
   }

public int getAvailableVolume(){return availableCapacity;}
}
```

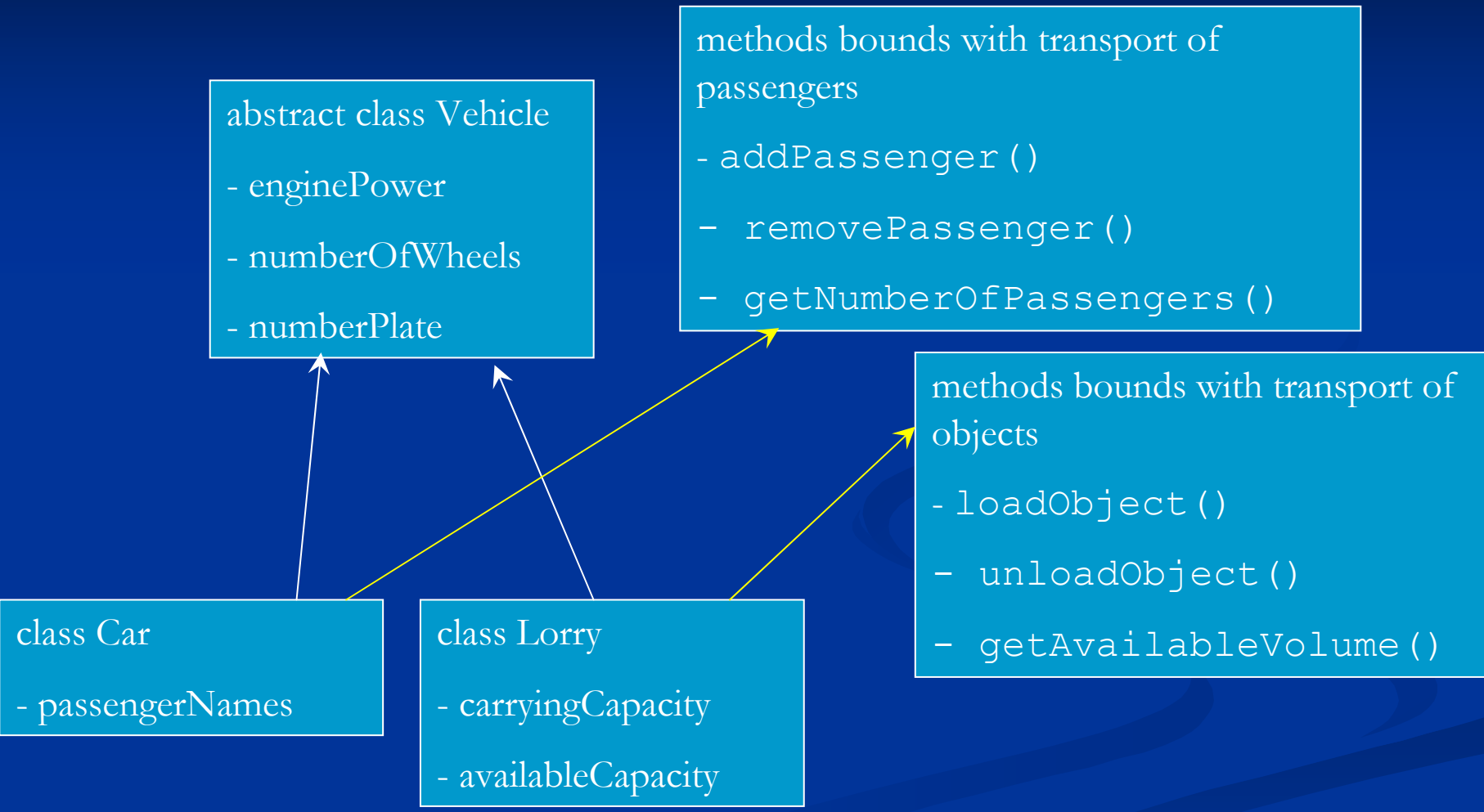
Why is this solution a bad one ?

- because attributes are duplicated

```
private int enginePower;
private int numberOfWheels;
private String numberPlate;
```

- because there is no standardization of the names of methods : buses would have to add or remove passengers too, planes would have to carry heavy objects too...

Building a better solution



Why better ?

- no duplication of source code
- smaller classes…easier to maintain, validate, look at…with local problems handled locally…
- a standardized behaviour
  - no trouble for users,
  - the capability to verify that objects used to do something were really designed to do it

This Solution in Java (1)

```
class Vehicle{
private int enginePower;
private int numberOfWheels;
private String numberPlate;

public void setPower(int somePower){enginePower=somePower;}
public int getPower(){return enginePower;}

public void setWheelsNumber(int number){numberOfWheels=number;}
public int getWheelsNumber(){return numberOfWheels;}

public void setNumberPlate(String aString){numberPlate=aString;}
public String getNumberPlate(){return numberPlate;}
}
```

This Solution in Java (2)

```
interface AbleToCarryPassengers{
public void addPassenger(String
  hisName);
public void removePassenger(int
  hisNumber)
  throws
  ArrayIndexOutOfBoundsException;
public int getNumberOfPassengers();
}
```

This Solution in Java (3)

```
import java.util.Vector;

class Car extends Vehicle implements AbleToCarryPassengers {
private Vector passengerNames;

public void addPassenger(String hisName){
    if (passengerNames == null) passengerNames=new Vector();
    passengerNames.add(hisName);
    }

public void removePassenger(int hisNumber) throws
    ArrayIndexOutOfBoundsException{
    if (passengerNames==null) throw new ArrayIndexOutOfBoundsException();
    passengerNames.remove(hisNumber);
    }

public int getNumberOfPassengers(){
    if (passengerNames==null) return 0;
    return passengerNames.size();
    }
}
```

This Solution in Java (4)

```
interface AbleToCarryObjects{
public void loadObject(int
  itsVolume) throws Exception;
public void unloadObject(int
  itsVolume) throws Exception;
public int getAvailableVolume();
}
```

This Solution in Java (5)

```
import java.util.Vector;

class Lorry extends Vehicle implements AbleToCarryObjects {
private int carryingCapacity, availableCapacity;

public void loadObject(int itsVolume) throws Exception{
    if (itsVolume <= 0) throw new Exception("BUG");
    if (itsVolume > availableCapacity) throw new Exception("Object can't be
    loaded");
    availableCapacity -= itsVolume;
    }

public void unloadObject(int itsVolume) throws Exception{
    if (itsVolume <= 0 || itsVolume > carryingCapacity-availableCapacity)
    throw new Exception("BUG");
    availableCapacity += itsVolume;
    }

public int getAvailableVolume(){return availableCapacity;}
}
```

This solution in C++(1)

```
#ifndef VEHICLE_H
#define VEHICLE_H
#include <string>

class Vehicle{
private :
int enginePower;
int numberOfWheels;
string numberPlate;
public :
void setPower(int somePower){enginePower=somePower;}
int getPower() const {return enginePower;}

void setWheelsNumber(int number){numberOfWheels=number;}
int getWheelsNumber() const {return numberOfWheels;}

void setNumberPlate(string aString){numberPlate=aString;}
const string getNumberPlate() const {return numberPlate;}
};
#endif
```

Necessary, because this file will probably be included by several other files

This solution in C++(2)

```
#include <string>

class AbleToCarryPassengers{
public :
virtual void addPassenger(const string hisName)=0;
virtual void removePassenger(int hisNumber)=0;
virtual int getNumberOfPassengers()const=0;
};
```

This solution in C++(3)

```
class AbleToCarryObjects{
public :
virtual void loadObject(int itsVolume)=0;
virtual void unloadObject(int itsVolume)=0;
virtual int getAvailableVolume()=0;
};
```

This solution in C++(4)

```
#include <vector>
#include <string>
#include "Vehicle.h"
#include "AbleToCarryPassenge
```

to allow public access to items in the class Vehicle if they are public, when using objects of Car class…

```
class Car : public Vehicle, AbleToCarryPassengers{
private :
vector <string> passengerNames;
public :
virtual void addPassenger(string hisName){
   passengerNames.push_back(hisName);
   }
```

because C++ will not search automatically for files…

This solution in C++(5)

```
  virtual void removePassenger(int
  hisNumber){
  vector<string>::iterator anIterator =
  passengerNames.begin();
  for (int i=0; i<hisNumber; i++)
  anIterator++;
  passengerNames.erase(anIterator);
  }
virtual int getNumberOfPassengers()const{
  return passengerNames.size();
  }
};
```

This solution in C++(6)

```
#include "Vehicle.h"
#include "AbleToCarryObjects.h"

class Lorry : public Vehicle, AbleToCarryObjects{
int carryingCapacity, availableCapacity;

public :
void loadObject(int itsVolume){
    if (itsVolume <= 0) throw "BUG";
    if (itsVolume > availableCapacity) throw "Object can't be loaded";
    availableCapacity -= itsVolume;
    }

void unloadObject(int itsVolume){
    if (itsVolume <= 0 || itsVolume > carryingCapacity-availableCapacity)
    throw "BUG";
    availableCapacity += itsVolume;
    }

int getAvailableVolume(){return availableCapacity;}
};
```

Automatically private

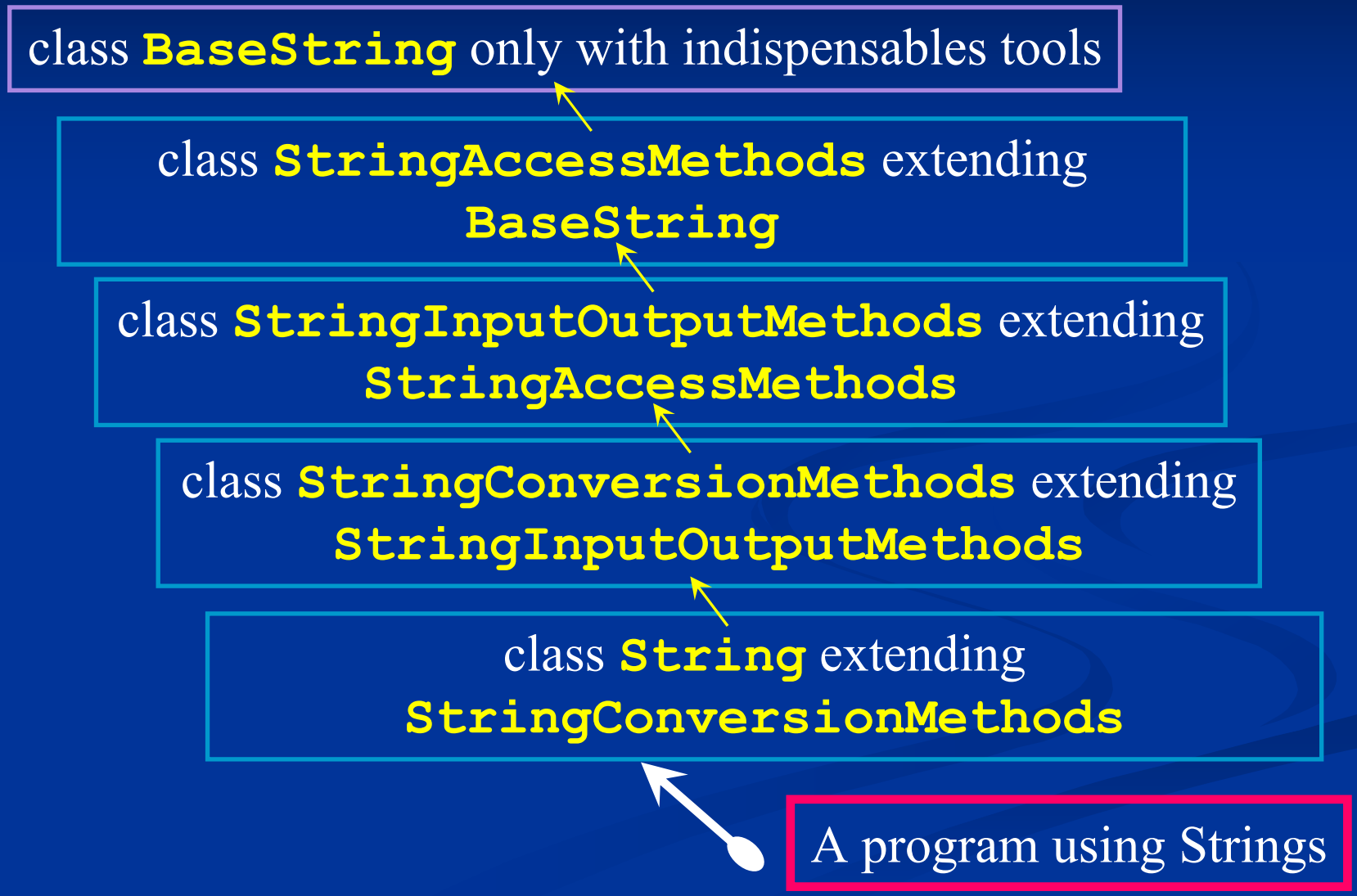
Organizing classes

- each class holds a set of attributes and a set of methods
- it's always easier to have small classes
  - easier to read,
  - easier to understand,
  - easier to maintain,
  - …
- derivation allows to build slowly the final class you need, step by step

Example of class string…

- This class needs methods to :
  - initialize, terminate,
  - handle input and output,
  - handle the size (increase size, decrease size, ask for size,...),
  - give access to characters (operators [ ], inserting,...),
  - compare (<, <=, ==, ...),
  - convert,
  - rectify, etc.
- The class also needs attributes, to store data (where the characters in memory are, what the current size is, what the max is. size if any, a.s.o.

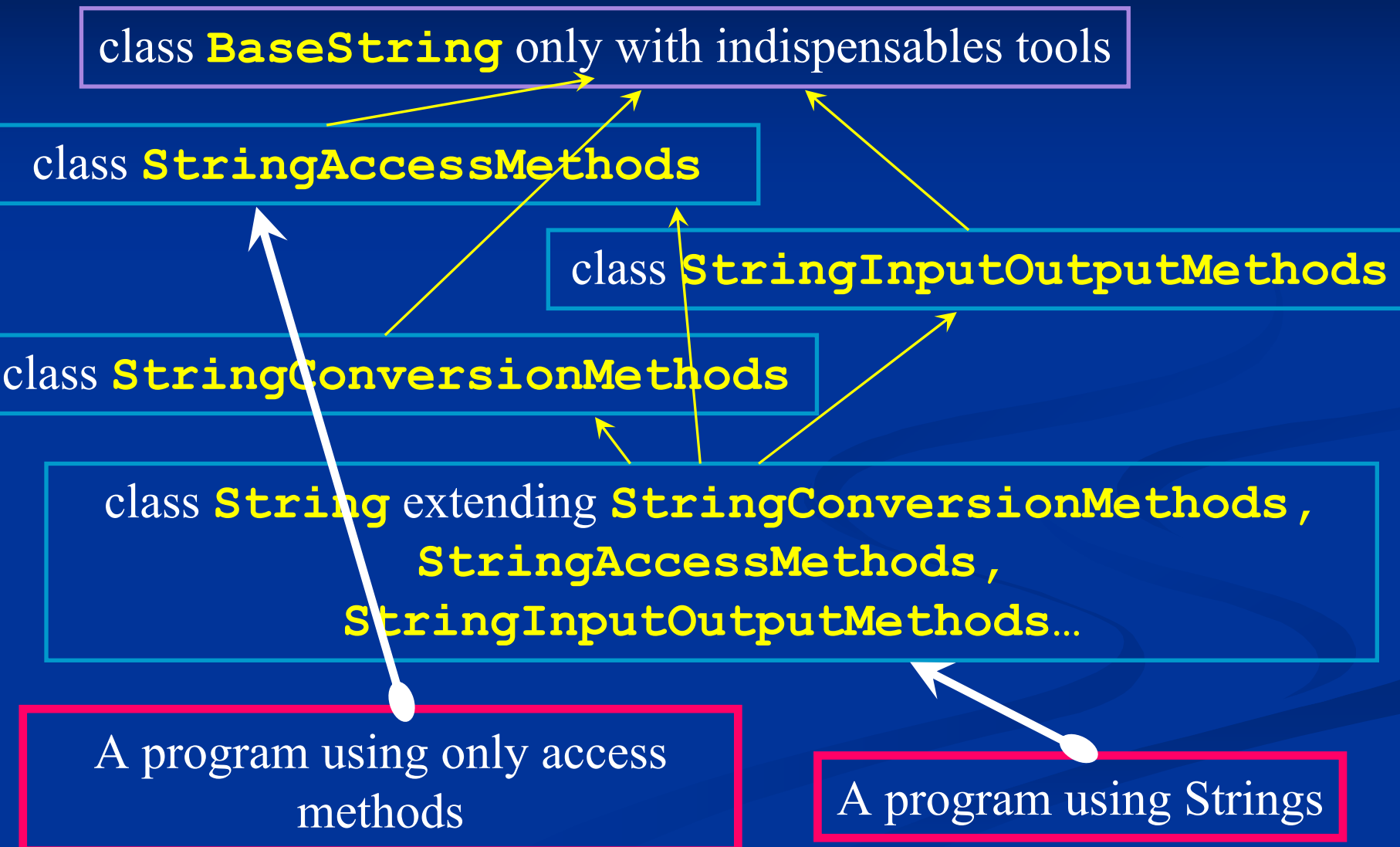
A Java solution



The Java solution

- allows to have small classes, each class taking care of a small set of methods
- allows to have the "end class" collecting all attributes and available methods
- forces the programmer to have at his disposal a lot of methods he doesn't need…

A C++ solution

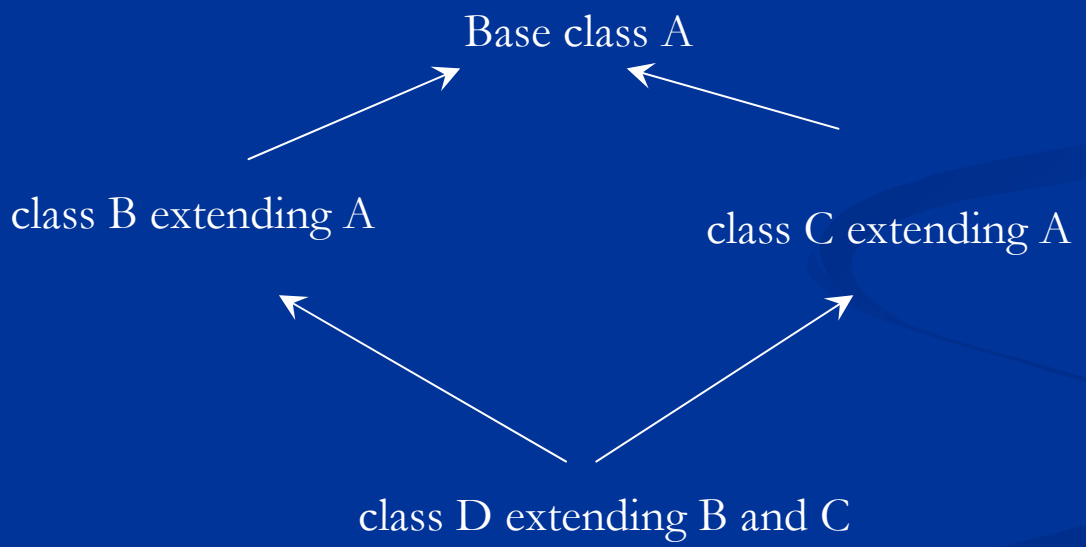

class BaseString only with indispensables tools
class StringAccessMethods
class StringInputOutputMethods
class StringConversionMethods
class String extending StringConversionMethods, StringAccessMethods, StringInputOutputMethods...
A program using only access methods
A program using Strings

The C++ solution

- allows to have small classes, each class taking care of a small set of methods
- allows to have the "end class" collecting all attributes and available methods
- allows the programmer to customize the set of classes he needs
- could give coherence problems…

Differences about derivation

- Java doesn't allow multiple derivations of objects => no diamond problem



In both languages

- each attribute or method may be either
  - **`private`** (may be used only inside of the class) or
  - **`protected`** (may be used inside of the class and in derived classes) or
  - **`public`** (may be used everywhere)
- but default values are different (private in C++, package public in Java)

Differences about derivation

- C++ knows derivation modes :
  - class B : protected A{…}
    - to forbid to the end user of class B (someone writing a program using objects of class B) to call methods of class A if they were not surcharged in class B…
  - class B : private A {…}
    - to forbid the calling of methods written in the A class or above when using objects of the B class or when writing classes deriving of the class B.

Differences about derivation

- C++ allows surcharging of operators (all known operators)
- Java doesn't :

```
Integer x = new Integer(5);
x = new Integer(x.intValue()+1);
```

- C++ would allow `x++`
- `int x = 5; x++;` is possible in both languages, but you sometimes need to store integer values as objects, for example in vectors…

About generic development

- suppose you have to write an algorithm searching some particular data among a set of data…

```
class Searching{

static boolean search(int [] anArray, int aValue){
   return search(anArray, aValue, 0);
   }

static boolean search(int [] anArray, int aValue, int index){
   if (index >= anArray.length) return false;
   return anArray[index]==aValue || search(anArray, aValue, index+1);
   }

public static void main(String[] args){
   int [] myArray = new int[args.length];
   for (int i=0; i<args.length; i++)
     myArray[i]=Integer.parseInt(args[i]);
   System.out.println("is 5 present in the given values ?
   "+search(myArray, 5));
   }
}
```

Never duplicate

```
static boolean search(int [] anArray, int aValue){
    return search(anArray, aValue, 0);
    }

static boolean search(int [] anArray, int aValue, int index){
    if (index >= anArray.length) return false;
    return anArray[index]==aValue || search(anArray, aValue, index+1);
    }

static boolean search(float [] anArray, float aValue){
    return search(anArray, aValue, 0);
    }

static boolean search(float [] anArray, float aValue, int index){
    if (index >= anArray.length) return false;
    return anArray[index]==aValue || search(anArray, aValue, index+1);
    }
```

About Duplicating algorithms

- duplicated algorithms take more place
- duplicated algorithms have to be tested separately
- algorithms may evolve (no guarantee that all copies will evolve together)

The Java solution : use of Objects

```
class SearchingWO{

static boolean search(Object [] anArray, Object aValue){
    return search(anArray, aValue, 0);
    }

static boolean search(Object [] anArray, Object aValue, int index){
    if (index >= anArray.length) return false;
    return anArray[index].equals(aValue) || search(anArray, aValue,
    index+1);
    }

public static void main(String[] args){
    Integer [] myArray = new Integer[args.length];
    for (int i=0; i<args.length; i++)
         myArray[i]=new Integer(Integer.parseInt(args[i]));
    System.out.println("is 5 present in the given values ? "+search(myArray,
    new Integer(5)));
    }
}
```

With C++

```
#include <iostream>
#include <stdlib.h>

bool search(int * anArray, int size, int aValue, int index){
    if (index >= size) return false;
    return anArray[index]==aValue || search(anArray, size, aValue, index+1);
    }

bool search(int * anArray, int size, int aValue){
    return search(anArray, size, aValue, 0);
    }

void main(int argc, char ** argv){
    int * myArray = new int[argc];
    for (int i=0; i<argc; i++)
         myArray[i]=atoi(argv[i]);
    cout << "is 5 present in the given values ? " << (search(myArray, argc,
    5)?"true":"false") << endl;
    }
```

With C++ and gene

will work with all objects belonging to classes having a good semantic for this operator

```
#include <iostream>
#include <stdlib.h>

template <class SomeType>
bool search(SomeType * anArray, int size, SomeType aValue, int index){
    if (index >= size) return false;
    return anArray[index]==aValue || search(anArray, size, aValue, index+1);
    }

template <class SomeType>
bool search(SomeType * anArray, int size, SomeType aValue){
    return search(anArray, size, aValue, 0);
    }

void main(int argc, char ** argv){
    int * myArray = new int[argc];
    for (int i=0; i<argc; i++)
         myArray[i]=atoi(argv[i]);
    cout << "is 5 present in the given values ? " << (search(myArray, argc,
    5)?"true":"false") << endl;
    }
```

About libraries

- standard libraries
  - Java SDK
  - C++ STL
- graphical libraries
  - Java => awt, swing…
  - C++ => TCL, openGL…
- Differences on software architecture
  - C++ doesn't need to use classes
  - packages and namespaces
  - **`makefile`**

Advantages of Java

- code is easier to read or to understand
- programs may be used in browsers, cellular phones, PDAs (pocket PCs)…
- Java makes it easy to build human-machine interfaces
- its automatic garbage collector

Advantages of C++

- real generic approach
- allowing a natural construction of hierarchy of classes (multiple inheritance)
- no hidden pointers
- a lot of details

Thank you !

http://www.iut-orsay.fr/~fournier

http://www.iffifs.u-psud.fr