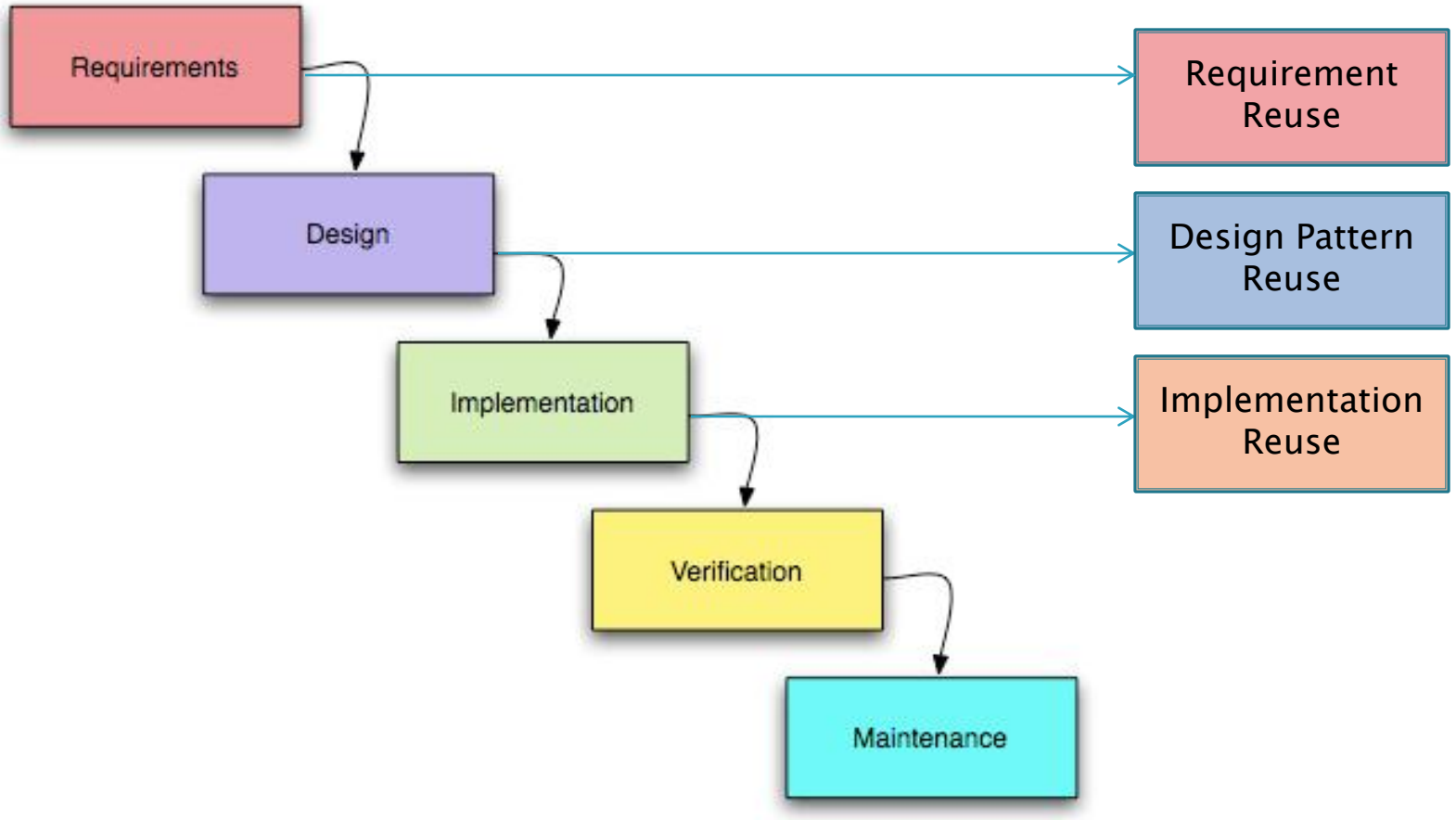
Module 4 Implementation Reuse (Part 1)

Dr. Shiping Chen

Where Are We?



Outline

- Procedures Reuse
- Classes(Object) Ruse
- System Calls Reuse

What is a Procedure?

- A portion of *code* within a larger program that performs a specific *task* and is relatively *independent* of the remaining code

- Also called **subroutine**, **function** or **subprogram**
- Usually written in C, Fortran, Pascal, PL/SQL etc.
- To complete a task, such as:
  - Mathematic Operations (matrix multiplication)
  - Image Processing (resize, color filtering)
  - Algorithms (Sorting, SVM)
- Usually packed as software library for reuse

An example of reusing procedure

Resize

The resize procedure in C

```
int enlarge_or_reduce(imgdes *image1, int pct)
{
  imgdes timage;
  int dx, dy, rcode;

  // Allocate space for the new image
  dx = (int)(((long)(image1->endx - image1->stx + 1)) * pct / 100);
  dy = (int)(((long)(image1->endy - image1->sty + 1)) * pct / 100);
  if((rcode = allocimage(&timage, dx, dy,
    image1->bmh->biBitCount)) == NO_ERROR) {
    // Resize Image into timage
    if((rcode = resizeex(image1, &timage, 1)) == NO_ERROR) {
      // Success, free source image
      freeimage(image1);
      // Assign timage to image1
      copyimgdes(&timage, image1);
      }
    else // Error in resizing image, release timage memory
      freeimage(&timage);
    }
  return(rcode);
}
```

Tips for Procedure-based SR

- When need to perform *common tasks* in your software system, don't rush to code!
- Instead, do some studies to see if any software library/packages provide the functionality that you need
- Once find one or more, you must carefully:
  - Read its manual, esp. its API and *prerequisites*
  - *Test it* with small & simple examples
  - Check its *license*, i.e. if there are any restrictions that prevent from your use in some cases!!!

Class in OOP

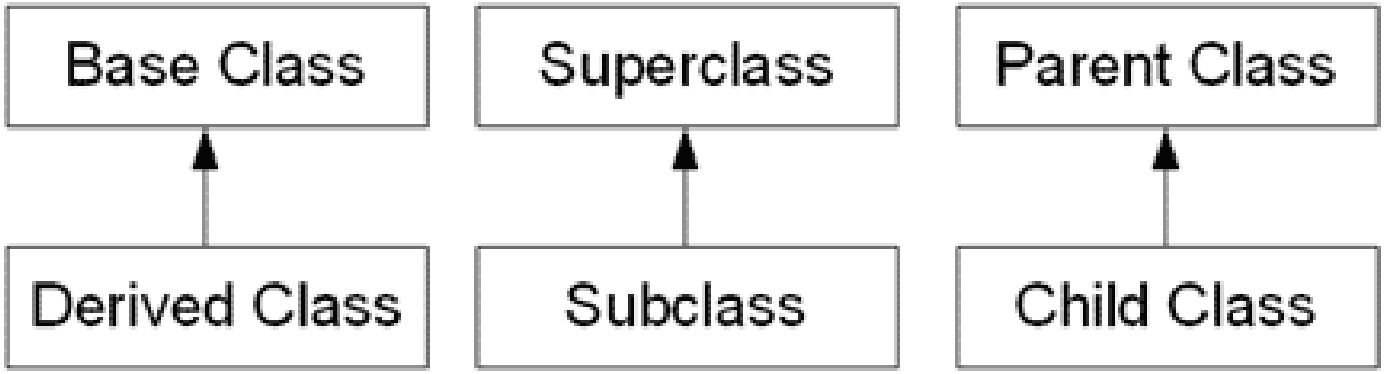
- Class is an abstraction of a particular type of objects in real life and/or the software world
- It captures the common structures/attributes and behaviors of the type of objects
- In OOP, it consists of:
  - public/private data members (attributes)
  - public/private methods (behaviors)
- It provides a great source for software reuse!

Class-Based Reuse

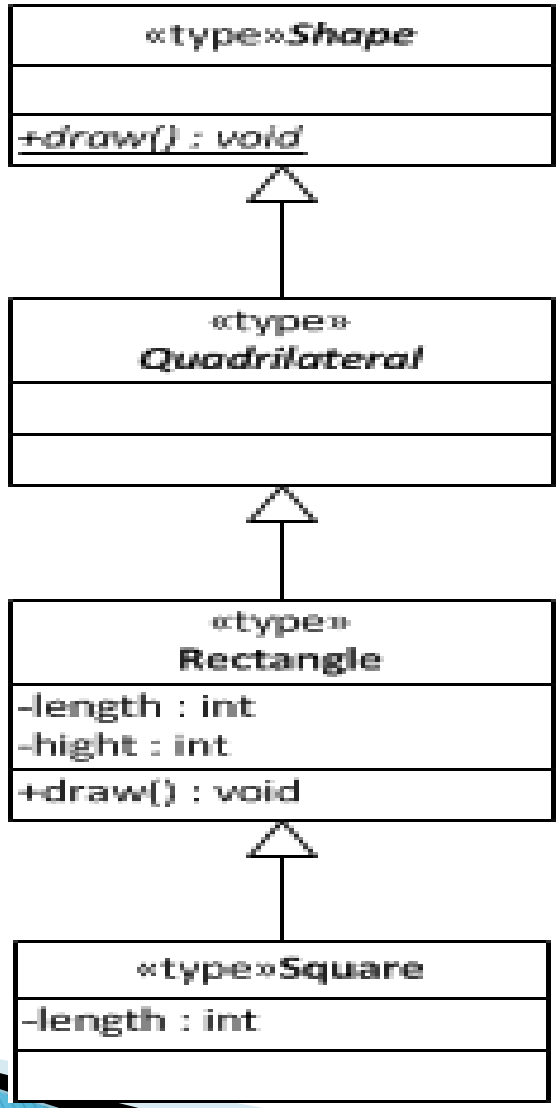
- 2 types of class-based reuse:
  - Inheritance
  - Composition

Class Inheritance

- In OOP, inheritance is a way to *reuse* code of existing class, establish a subtype from 1 or more existing class



An Example of Class Inheritance




```
Class Square {
    ......
    Class Rectangle {
        ......
        Class Quadrilateral {
            ......
            Class Shape {
                ......
```

The Classes in Java

Shape.java

```
public abstract class Shape
{
        // declare abstract methods
        abstract void draw();
}
```

The Classes in Java

Rectangle.java	Square.java

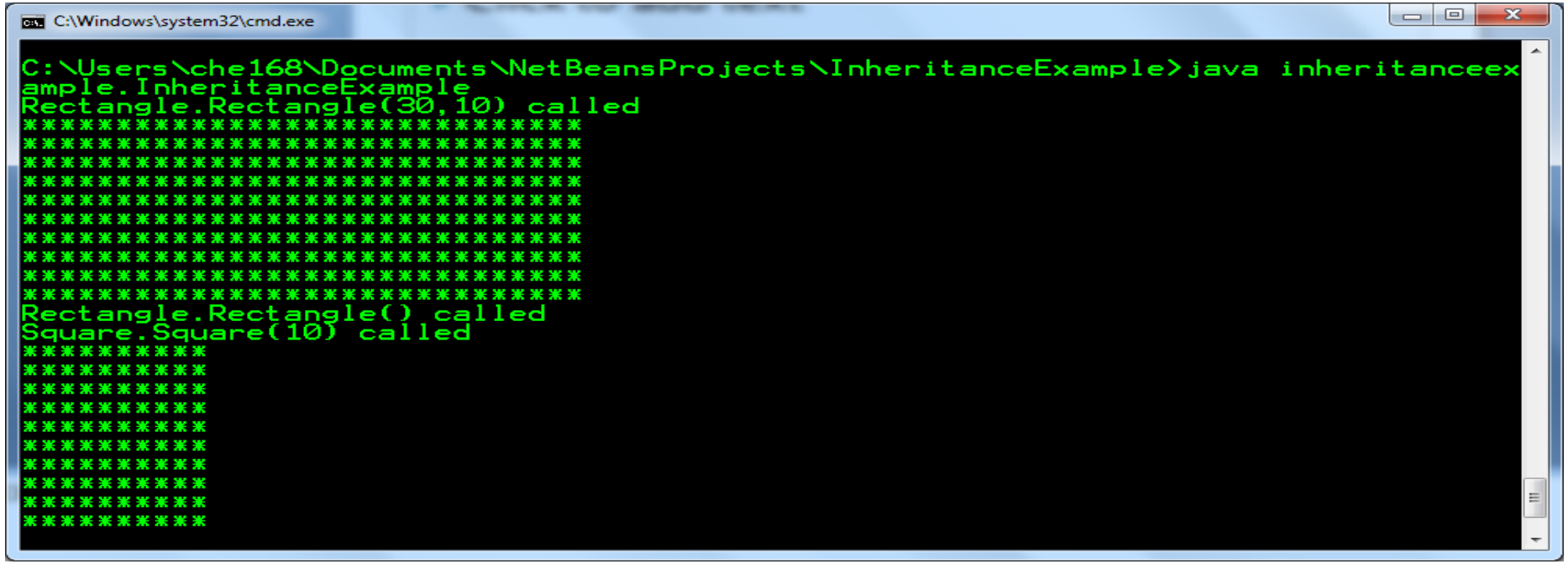
public class Rectangle extends Shape {
  int length, height;
 	
  public Rectangle() {}
  public Rectangle(int length, int height) {
  this.length = length;
  this.height = height;
  }
 	
  @Override
  void draw() {
  for(int i=0; i<this.height; ++i) {
  for(int j=0; j<this.length; ++j){
  System.out.print("*");
  }
  System.out.println();
  }
  }
}
	public class Square extends Rectangle
	{
	  public Square(int length)
	  {
	  this.length = length;
	  this.hight = length;
	  }
	}

Let us test them

Tester.java

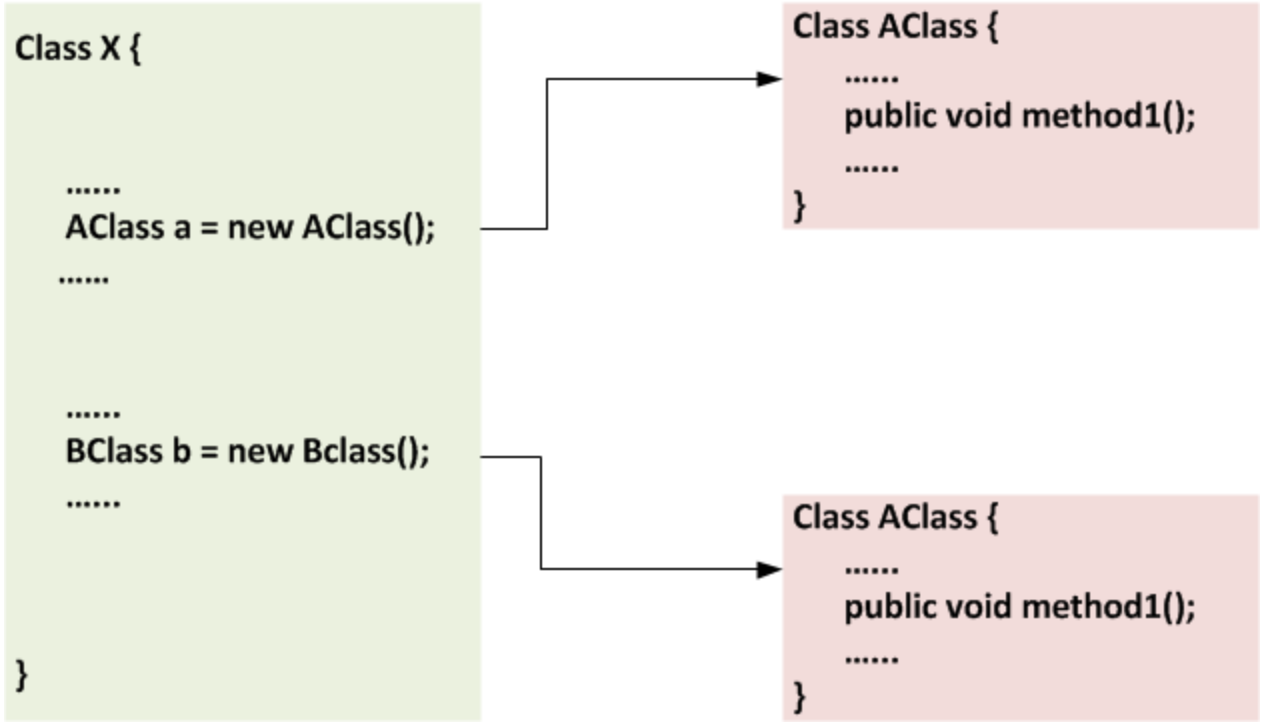
```
public class Tester
{
      public static void main(String[] args)
     {
            Rectangle rect = new Rectangle(30, 10);
            rect.draw();

            Square square = new Square(10);
            square.draw();
   }
}
```

Here is what we get

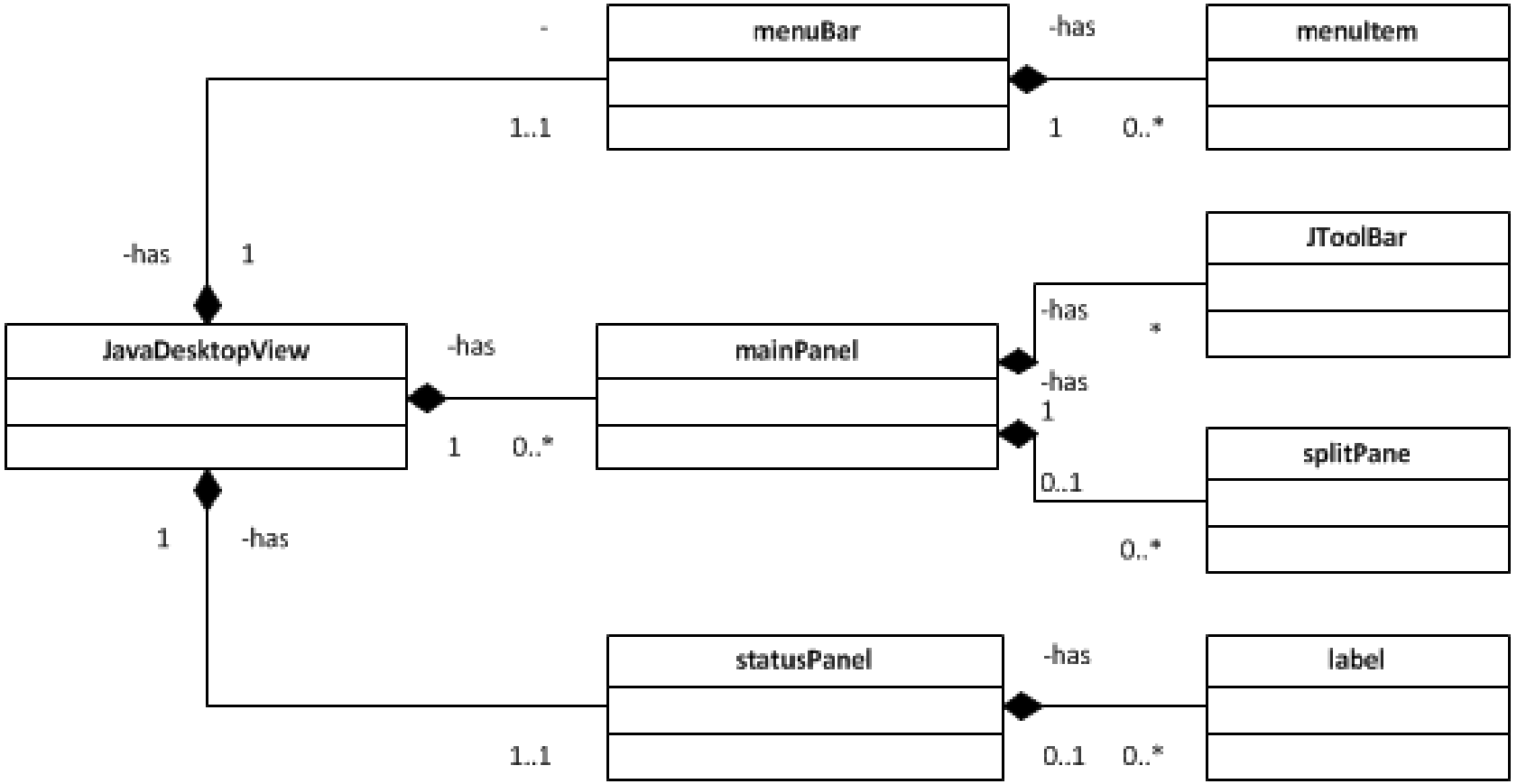


Class Composition

- An alternative way to reuse the implementation of other classes



An Example of Class Composition

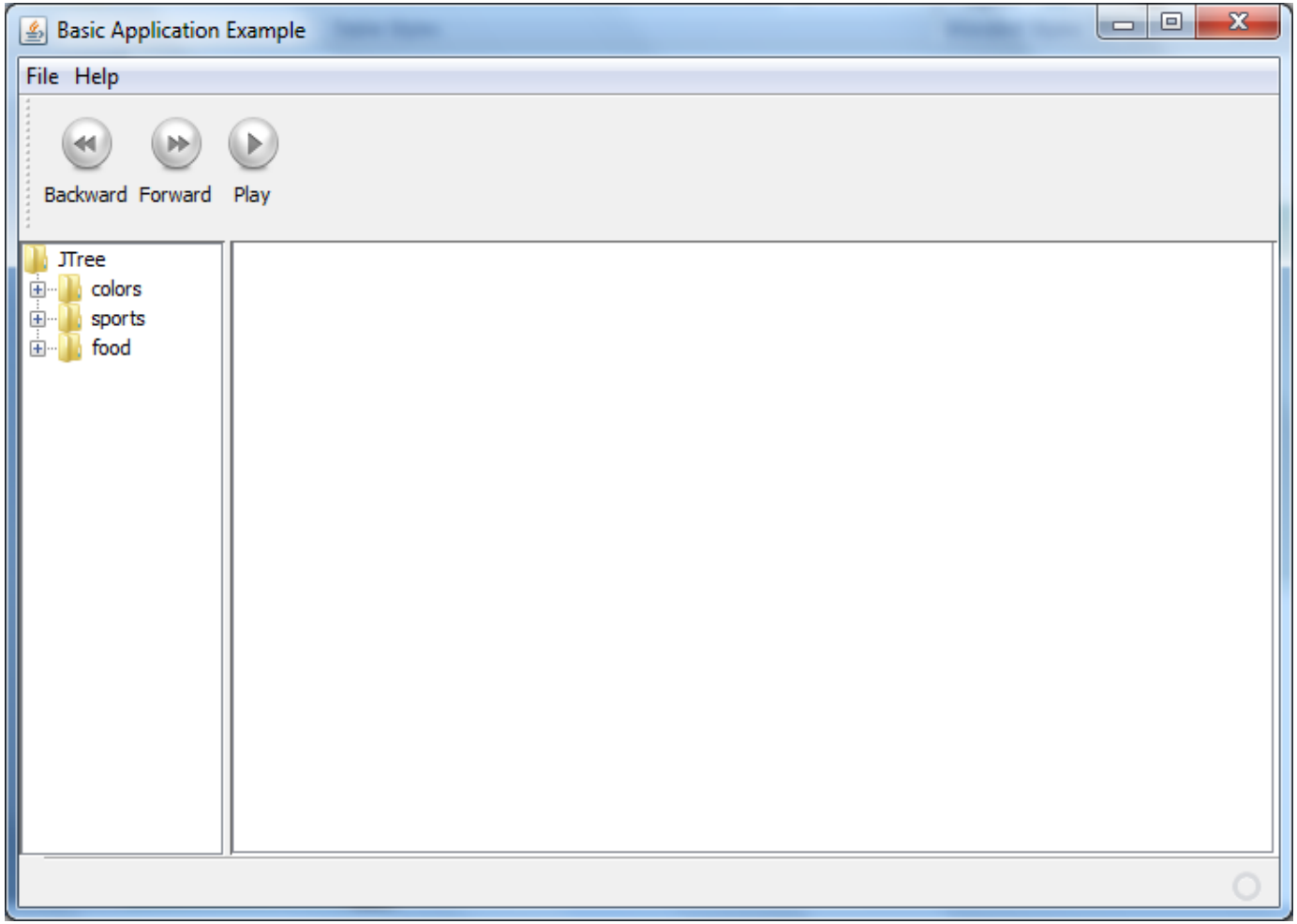


The Class in Java

JavaDesktopView.java

```
public class JavaDesktopView extends FrameView {
    private javax.swing.JMenuBar menuBar;
    private javax.swing.JToolBar jToolBar1;
    private javax.swing.JPanel mainPanel;
    private javax.swing.JPanel statusPanel;
    ……
    private void initComponents() {
        menuBar.add(fileMenu);
        setMenuBar(menuBar);

        ……
        setStatusBar(statusPanel);
    }
}
```

To run the example



Tips to write reusable code/class

- Avoid repeat
- Make one class/method do one thing
- Write unit test for your class
- Remove business logic and main code
- Try to think/use abstract classes or interfaces
- Code for extension
- Don’t write code that you don’t need
- Try to reduce coupling

System Call Reuse

- System Call
  - A program requests a service from an Operating System (OS)'s kernel
  - the interface between a process and the operating system
- Another source for software reuse!

An example of system call reuse

Requirements:

- In most java programs, java developers use the following java class to measure time:

```
static long System.currentTimeMillis();
```

- For example,

```
long t1 = System.currentTimeMillis();
..................... // do something
long t2 = System.currentTimeMillis();
long RS = t2 – t1
```

Issue/Problem of using Java System.currentTimeMillis();

- The resolution of System.currentTimeMillis(); is about ~20 msec.
- It is OK for measuring most systems;
- But it is not OK if measuring very fast systems!

Solution: High Resolution Timer Using System Call

- Can improve resolution over 1000%, i.e. *macro second*
- **Using system call, i.e. on windows:**
  - QueryPerformanceFrequency(out freq)
  - QueryPerformanceCounter(out tick);
- **Using JNI technology – Java Native Interface**

A High Resolution Timer in Java

```
class HiResTimer
{
   public native boolean isHighResTimerAvailable();
   public native double  getResolution();
   public native double  startTiming();
   public native double  endTiming(double dStart );
}
```

HiResTimer.cpp

```
#include <windows.h>
#include <com_shiping_util_HiResTimer.h>

// ----------------------------
// HiResCounter
// ----------------------------
class _HiResCounter {
  public:
    _HiResCounter()
    {
        m_bExists = QueryPerformanceFrequency(
       &m_Freq );
     }

  BOOL Exists() { return m_bExists; }
  double Freq() { return (double)(m_Freq.QuadPart); }

  protected:
    BOOL m_bExists;
    LARGE_INTEGER m_Freq;
};
```

HiResTimer.cpp (Con't)

```
static _HiResCounter HiResCounter;

// ------------------------------------------
// Class:     HiResTimer
// Method:    isHighResTimerAvailable
// Signature: ()Z
// ------------------------------------------
JNIEXPORT jboolean JNICALL
Java_com_shiping_util_HiResTimer_isHighResTimerAvailable
   (JNIEnv *, jobject)
{
   return HiResCounter.Exists();
}

.........
```

Summary of Part 1

We learnt:

- How to reuse procedures
- How to reuse classes
  - Inheritance
  - Composition
- How to reuse system calls