CSUS
COLLEGE OF ENGINEERING AND COMPUTER SCIENCE
Department of Computer Science

CSc 133 — Object-Oriented Graphics Programming (Fall 2018)

Name Minquan Li

MIDTERM EXAM

1. Write your name in the space above.
2. The exam is closed book, closed notes, except that you may use a *single sheet* of *hand-written* notes. If you use a note sheet you must put your name on it and turn it in with your exam.
3. There are 100 total points; you have 60 minutes to work on it — budget your time accordingly.
4. Absolutely NO use of ANY electronic devices is allowed during the exam. This includes cell phones, tablets, laptops, or any other communications device.
5. Please be neat — I cannot give credit to answers I cannot read.
6. The exam has 12 pages (including the work space), counting this cover page. Make sure you have all the pages.

Problem	Points	Possible
1	34	50
2	10	15
3	0	15
4	13	20
Total	57	100

1. Multiple Choice/Selection, Short Answers. Write the letter of the best answer in the blank to the left.

D A certain Java/CN1 class named "**Point**" has constructors "**Point()**" and "**Point(int x, int y)**". This is an example of

A. abstraction B. encapsulation C. inheritance D. overloading E. overriding

_N/A_ Give an example, each, of an Overloading and Overriding in CN1/Java

Overloading: Public class A { print (int num) {...} print (String str) {...} }

Overriding: public class B extends A { @Override print (int num) {...} }

D Which of the following class relationships best fits the composite pattern?

A. Zoo contains a Set, an Exhibit contains a Set, and an Animal contains a number of properties about that individual animal. To get information about a particular Animal, a client would write something such as:
Zoo.getExhibit("Penguins").getPenguin("Tux").getAge();

B. Dalmatian is a subclass of Dog, which is a subclass of Mammal, which is a subclass of Animal. Each subclass overrides some methods while using the inherited version of others, for some shared behavior and some distinct behavior.

C. GeometricShape is an interface implemented by Square, Circle, Sphere, and Dodecahedron. Though they have the same public interface and can all be used anywhere a GeometricShape is required, they otherwise have no relationship and do not depend on each other.

D. The class Food is implemented by PeanutButterAndJellySandwich, which contains objects of type Bread, PeanutButter, and Jelly. Bread contains Flour and Salt, and Jelly contains Fruit and Sugar. All of these objects are Food objects themselves.

E. None of the above

B A certain Java/CN1 class named "B" extends another class named "A". Class B defines a method named "C" with the same signature as that of a method named "C" in Class A. Method C does not contain the keyword "super". A program constructs an instance of B and invokes method "C" in that object. The code which will be executed as a result of this invocation is

A. the code in A.C
B. the code in B.C
C. the code in A.C followed by the code in B.C
D. the code in B.C followed by the code in A.C
E. it depends on the code in A.C
F. it depends on the code in B.C
G. None of the above

B The Output of the following code is correct:

```
class Vehicle {
  public void applyBrakes() {
    System.out.printf ("Applying vehicle brakes\n");
  }
}
class Truck extends Vehicle {
  public void applyBrakes() {
    System.out.printf("Applying truck brakes...\n");
  }
}
```

```
public static void main (String [] args){
  Vehicle v;
  Truck t;
  t = new Truck();
  t.applyBrakes();
  v = t ;
  v.applyBrakes();
}
```

Output

```
Applying vehicle brakes...
Applying truck brakes...
```

A. True.
B. False.

D When would you use a private constructor?

A. When you get bored with public
B. If you want to disallow instantiation of that class from outside that class
C. If you want to protect your class's members from outside modification
D. Never, it's not allowed

C If a Java/CN1 program contains a declaration such as "class A {…}", where "…" represents the code defining the class, then

A. A has no parent class
B. A is its own parent
C. A is a superclass of Object
D. A is an abstraction of Object
E. A is a subclass of Object

D Multiple inheritance leaves room for a derived class to have _____ members.

A. Private
B. Public
C. Ambiguous
D. Protected

_N/A_ Explain what is Polymorphism in context of CN1/Java? How this concept is being utilized in your assignment 1? Motivate your answer with a short of snippet code.

-1 **Polymorphism:** polymorphic reference can refer to different object types at runtime. Inheritance, method overriding, polymorphic assignment, and ploymorphic methods are examples of Polymorphism.

-1 **Example:** Public class Hello extends World {...}
or Public class A { print (int a) {...}
print (String s) {...} }

-8

Q: When would you use a private constructor?

A: (B). A private constructor will only allow the class to instantiate from within the class and not from outside.

Q: If a Java/CN1 program contains a declaration such as “class A {…}”, where “…” represents the code defining the class, then

A: (E). There is a cosmic base class called Object Every class except Object is a direct or indirect subclass of Object. Class A has no parent, so it is a subclass of Object.

Q: Multiple inheritance leaves room for a derived class to have _____ members.

A: (C). Inheritance lets the child class to keep all of the parent functions, but at the same time the child class can do different things and be ambiguous.

Q: Explain what is Polymorphism in context of CN1/Java? How this concept is being utilized in your assignment 1? Motivate your answer with a short of snippet code.

A: **Polymorphism:** In CN1/Java it means that you can do operations with different types of objects or the operations can be done in more than one ways.

A: **Example:** In assignment 1, all the objects are stored in an array. However, we can check if each object is an instance of MoveableGameObject and call the move() function safely in a polymorphic manner.

```
for (int i=0; i<store.size(); i++)
{
    if (store.elementAt(i) instanceof MoveableGameObject)
    {
        MoveableGameObject mgObj = (MoveableGameObject) store.elementAt(i);
        mgObj.move();
    }
```

A ____ A certain Java/CN1 class named `Sphere` contains a method named `getColor()` which returns the color of the `Sphere` object. This method is an example of a (an)

A. accessor B. mutator C. aggregation D. design pattern E. abstraction

C ____ Which of the following describes the Singleton pattern correctly?

A. This pattern creates object without exposing the creation logic to the client and refer to newly created object using a common interface.
B. In this pattern an interface is responsible for creating a factory of related objects without explicitly specifying their classes.
C. This pattern involves a single class which is responsible to create an object while making sure that only single object gets created.
D. This pattern is used when we want to pass data with multiple attributes in one shot from client to server.

C ____ Which of the following pattern is used when there is one-to-many relationship between objects such as if one object is modified, its dependent objects are to be notified automatically?

A. Iterator.
B. Factory.
C. Observer.
D. None of the above.

D ____ What must a non-abstract child do about an abstract method in its parent class?

A. A child must override an abstract method inherited from its parent by defining a method with the same signature and same return type.

B. A child must define an additional method similar to the one inherited from its parent by defining a method with the same signature and different return type.

C. A child must not define any method with the same signature as the parent's abstract method.

D. A non-abstract child must define an abstract method with the same signature and same return type as the parent's abstract method.

C ____ If the child object can exist beyond the lifetime of its parent, then the relationship is:

A. Generalization.
B. Composition.
C. Aggregation.
D. None of the above.

Q: What must a non-abstract child do about an abstract method n its parent class?

A: (A). An abstract method is a contract to make sure its concrete child will have the same method name and returns the same return type.

A What relationship is appropriate for a CSC-133 Course and a Student?

A. Association.
B. Aggregation.
C. Composition.
D. Dependency.
E. None of the above.

A Why would a class be declared as abstract?

A. Because it doesn't make logical sense to instantiate it.
B. So that it can be used as an interface.
C. So that it cannot be inherited from.
D. Because it has no abstract methods.
E. None of the above.

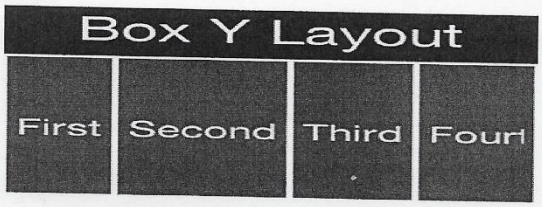
B What was it created from (never changes) determines: Which implementation to call when the method is invoked → Apparent Type

A. True.
B. False.

B The following code:

```
Form hi = new Form("Box Y Layout", new BoxLayout(BoxLayout.Y_AXIS));
hi.add(new Label("First")).
    add(new Label("Second")).
    add(new Label("Third")).
    add(new Label("Fourth")).
    add(new Label("Fifth"));
```

Which results in this:

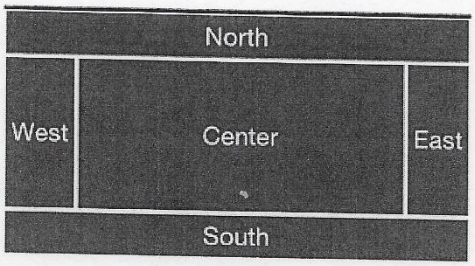


A. True.
B. False.

Q: What relationship is appropriate for a CSC-133 Course and a Student?

A: (B). The student is still alive even after CSC-133 is finished. It’s aggregation because the student can exist without the course.

B The following is referred to as "FlowLayout".



A. True.
B. False.

A Codename one application has the following lifecycle:

A. init/start/stop/destroy
B. constructor/destructor
C. start/go/stop
D. initialize/start/stop/return

A Which of the following describes the Structural pattern correctly?

A. This type of patterns provides a way to create objects while hiding the creation logic, rather than instantiating objects directly using new operator.
B. This type of patterns concerns class and object composition. Concept of inheritance is used to compose interfaces and define ways to compose objects to obtain new functionalities.
C. This type of pattern is specifically concerned with communication between objects.
D. This type of pattern is specifically concerned with the presentation tier.

D Which of the following characteristics of an object-oriented programming language restricts behavior so that an object can only perform actions that are defined for its class?

A. Dynamic Binding
B. Polymorphism
C. Inheritance
D. Encapsulation

Q: Which of the following describes the Structural pattern correctly?

A: (B). Structural design patterns are design patterns that ease the design by identifying a simple way to realize relationships among entities.

Provide the expected output for the following code:

```
class Animal {
   public void move() {
      System.out.println("Animals can move");
   }
}

class Dog extends Animal {
   public void move() {
      System.out.println("Dogs can walk and run");
   }
   public void bark() {
      System.out.println("Dogs can bark");
   }
}

public class TestDog {

   public static void main(String args[]) {
      Animal a = new Animal();   // Animal reference and object
      Animal b = new Dog();   // Animal reference but Dog object

      a.move();   // runs the method in Animal class
      b.move();   // runs the method in Dog class
      b.bark();
   }
}
```

Animal move()
↑
Dog @Override move() bark()

Output:
Animals can move
Dogs can walk and run
Dogs can bark

Selection: Place the following selection(s) (use Letter(s) Only, i.e A, B) into appropriate column(s). Use only features below Java 8 (Unchecked as in Codename One – Project)

Java Abstract	Java Interface
A	C
D	B
F	E

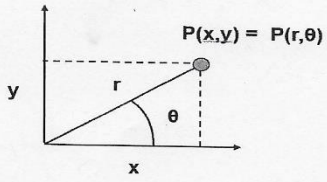
A. can have both abstract and concrete methods
B. supports multiple inheritance
C. can only have public static final (constant) variable
D. there is a clear inheritance hierarchy to be defined
E. if various implementations only share method signatures then it is better to use
F. contains constructor

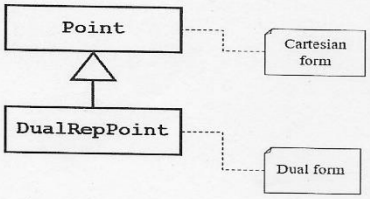
1) The parent type Animal does not know about the bark() method in Dog object. So b.bark(); will generate an error and output would be compilation error message.

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        The method bark() is undefined for the type Animal

        at TestDog.main(TestDog.java:30)
```

2. **Inheritance**: Consider the following DualRepPoint Class. It supports both Cartesian and Polar Coordinates.





You are given the following Classes:

```
public class Point {
        private double x, y ;
        public Point (){
                x = 0.0 ;
                y = 0.0 ;
        }

        public double getX() { return x ; }
        public double getY() { return y ; }
        public void setX (double newX) {
                x = newX ;
        }
        public void setY (double newY) {
                y = newY ;
        }
}
```

```
/** This class maintains a point representation in both Polar and
Rectangular form and protects against inconsistent changes in the local
fields */

public class DualRepPoint extends Point {
        private double radius, angle ;

        public DualRepPoint () {
                radius = 2.0 ;
                angle = 45.0;
                updateRectangularValues();
        }
        public double getRadius() { return radius ; }
        public double getAngle() { return angle ; }
        public void setRadius(double theRadius) {
                radius = theRadius ;
                updateRectangularValues();
        }
        public void setAngle(double angleInDegrees) {
                angle = angleInDegrees;
                updateRectangularValues();
        }
        // force the Cartesian values (inherited from Point)
        // to  be consistent
        private void updateRectangularValues() {
                x = radius * Math.cos(Math.toRadians(angle));
                y = radius * Math.sin(Math.toRadians(angle));
        }
}
```

can not reach private field of point class.
this.setX(radius * Math.cos(Math.toRadians(angle)));
this.setY(radius * Math.sin(Math.toRadians(angle)));

Here is a client class who would like to use the DualRepPoint :

```
public class SomeClientClass {
        private DualRepPoint myDRPoint ;
        public SomeClientClass() {        // client constructor
                myDRPoint = new DualRepPoint(); // create a private DualRepPoint
                myDRPoint.setX(2.2) ;
                myDRPoint.setY(7.7) ;
        }
}
```

DualRepPoint does not have these methods
wrong. these methods exist in DRP's parent

myDRPoint.setRadius(2.2); ×

Please analyze the above classes. (1) If there is an issue(s) with any of the class(es) above, please state the reason(s) and describe how to fix it (them). Or (2) Otherwise, if there is no issue with these classes, please provide your justifications.

```java
public class Point
{
    private double x, y;
    public Point()
    {
        x=0.0;
        y=0.0;
    }

    public double getX() {return x;}
    public double getY() {return y;}
    public void setX(double newX)
    {
        x = newX;
    }
    public void setY(double newY)
    {
        y = newY;
    }
}
```

```java
public class DualRepPoint extends Point{
    private double radius, angle;

    public DualRepPoint(){
        radius = 2.0;
        angle = 45.0;
        updateRectangularValues();
    }
    public double getRadius() {return radius;}
    public double getAngle() {return angle;}
    public void setRadius(double r){
        radius = r;
        updateRectangularValues();
    }
    public void setAngle(double a){
        angle = a;
        updateRectangularValues();
    }
    public void setX(double newX){
        super.setX(newX);
        updatePolarValues();
    }
    public void setY(double newY){
        super.setY(newY);
        updatePolarValues();
    }

    private void updatePolarValues(){
        double x = super.getX();
        double y = super.getY();
        radius = Math.sqrt( (x*x) + (y*y) );
        angle = Math.toDegrees(Math.atan2(y, x));
    }
    private void updateRectangularValues() {
        super.setX(radius*Math.cos(Math.toRadians(angle)));
        super.setY(radius*Math.sin(Math.toRadians(angle)));
    }
}
```

```java
public class SomeClientClass
{
    private DualRepPoint myDRPoint;
    public SomeClientClass()
    {
        myDRPoint = new DualRepPoint();
        myDRPoint.setX(2.2);
        myDRPoint.setY(7.7);
    }

    public static void main (String[]args)
    {
        SomeClientClass a = new SomeClientClass();
        System.out.println("Angle: "+a.myDRPoint.getAngle());
        System.out.println("Radius: "+a.myDRPoint.getRadius());
    }
}
```

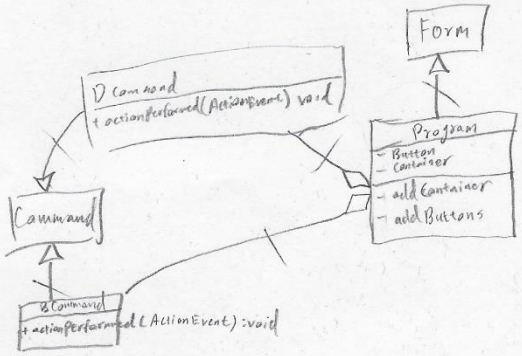
Markers Properties Servers Data Source Explorer Snippets Console

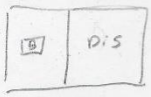
```
<terminated> SomeClientClass [Java Application] C:\Program Files\Java\jre1.8.0_191\bin\javaw.ex
Angle: 74.05460409907715
Radius: 8.00812087820857
```

Correction: MyDRPoint.setY and MyDRPoint.setX is legal because both methods are inherited from parent. Nothing is wrong with the Point class because it has private variables and has getters and setters to represent the cartesian form. However, the client should not be able to mess with parent's setters because it does not compute both the points and the angle.

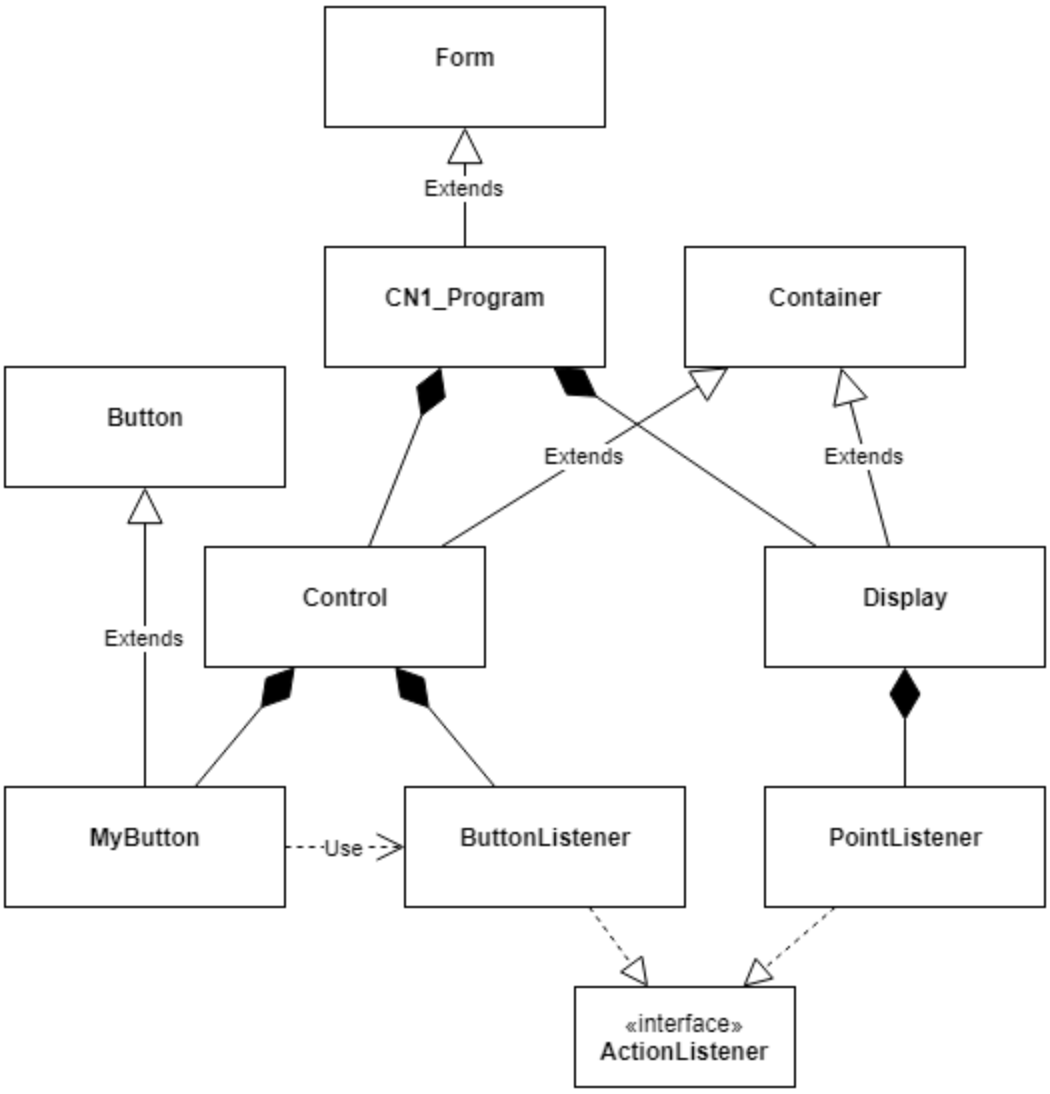
3. **Basic User Interface and UML class diagram**: An CN1 program displays a form which contains two components: a control container containing a single button, and a separate display container. The button has an action listener attached to it which is an instance of a separate class. The display container has a pointer listener attached to it which is likewise an instance of a (different) separate class. The form is a subclass of **`Form`**; the containers are subclasses of **`Container`**, and the button is a **`Button`**.

Draw a UML diagram depicting the associations between the elements of this program.

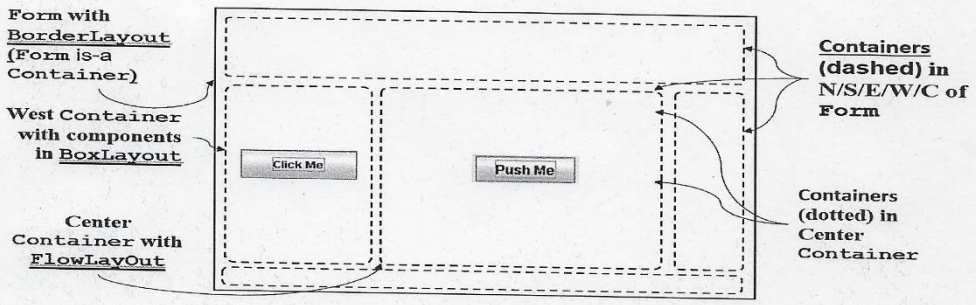




- 6: missing classes
- 7: incorrect relations
- 2: missing interfaces


Form
Extends
CN1_Program
Container
Button
Extends
Extends
Control
Display
Extends
MyButton
Use
ButtonListener
PointListener
«interface»
ActionListener

4. **Command design pattern:** Writing a complete CN1 program to solve the described problem below. The picture here is a simple Graphical User Interface (GUI) consisting of a form. It is configured with BorderLayout and having the following buttons: Button1 (Click Me) and Button2 (Push Me). The buttons are placed in designated West and Center containers respectively. The West Container is configured with BoxLayout and the Center Container is configured with FlowLayout.



The handling for the command via the buttons are satisfied by the System.out.println text displaying of which button invocation takes place (i.e. "Clicked from Click Me" vs. "Pushed from Push Me"). Additionally, user can select letter '**c**' to invoke the 'Click Me' command. User can select letter '**p**' to invoke the 'Push Me" command. You are required to use **ONLY Command Design Pattern** to solve this problem. Please provide a complete CN1/Java implementation for this problem.

```
public class GUI extends Form {
  this.setLayout(new BorderLayout());
  // west Container
  Container wContainer = new Container(); ✓
  Button cButton = new Button("Click Me"); ✓
  wContainer.setLayout(new BoxLayout.~~CENTER~~); ✓
  wContainer.add(cButton); ✓
  PrintCommand p = new PrintCommand(); ✓
  cButton.setCommand(P); ✓
  this.add(wContainer); ✗

  // center
  Container cContainer = new Container(); ✓  printCommand2 p2 = new PrintCommand2(); ✓
  Button pButton = new Button("Push Me"); ✓
  cContainer.add(pButton); ✓
  pButton.setCommand(P2); ✓
  this.add(cContainer); ✗

  // Key command
  this.addKeyListener('c', P); ✓
  this.addKeyListener('p', P2); ✓

}
```

-4: missing command classes.

```
import com.codename1.ui.Button;
import com.codename1.ui.Command;
import com.codename1.ui.Container;
import com.codename1.ui.Form;
import com.codename1.ui.events.ActionEvent;
import com.codename1.ui.layouts.BorderLayout;
import com.codename1.ui.layouts.BoxLayout;

class ClickPrintCommand extends Command
{
        public ClickPrintCommand(String command)
        {
                super(command);
        }

        public void actionPerformed(ActionEvent e)
        {
                System.out.println("Clicked.");
        }
}

class PushPrintCommand extends Command
{
        public PushPrintCommand(String command)
        {
                super(command);
        }

        public void actionPerformed(ActionEvent e)
        {
                System.out.println("Pushed.");
        }
}

public class GUI extends Form
{
        public GUI()
        {
                this.setLayout(new BorderLayout());
                // west container
                Container westContainer = new Container();
                Button clickButton = new Button();
                westContainer.setLayout(new BoxLayout(1));
                westContainer.add(clickButton);
                ClickPrintCommand c = new ClickPrintCommand("Click Me");
                clickButton.setCommand(c);
                this.add(BorderLayout.WEST, westContainer);
                // center container
                Container centerContainer = new Container();
                Button pushButton = new Button();
                centerContainer.add(pushButton);
                PushPrintCommand p = new PushPrintCommand("Push Me");
                pushButton.setCommand(p);
                this.add(BorderLayout.CENTER, centerContainer);
                // key command
                this.addKeyListener('c', c);
                this.addKeyListener('p', p);
                // show
                this.show();
        }
}
```

Output: