

Exercises related to Java and applets

Kari Laitinen

<http://www.naturalprogramming.com>

2005-06-07 File created.

2007-04-19 Last modification.

- When you work with JCreator, it is not necessary to create projects. You can simply open the .java files to the editor and compile them. When you need to execute an applet, you can open the necessary .html file and execute that. JCreator automatically starts the appletviewer program when you execute a .html file.
- When you develop Java programs, don't keep your .java files on a disk that is on the school network.

Note that these exercises are written for the students that I teach.

Java GUI programming (e.g. applets) are not covered in "A Natural Introduction to Computer Programming with Java"

First Java Exercises: A program to convert temperature values

There are two common systems for measuring temperature. Degrees of Fahrenheit (°F) are used in the U.S. and some other countries, while degrees of Celsius (°C) are in use in most European countries and in many countries throughout the world. The freezing point of water is 0 degrees Celsius and 32 degrees Fahrenheit, 10°C is 50°F, 20°C is 68°F, 30°C is 86°F, and so on. You can see that 10 degrees on the Celsius scale corresponds to 18 degrees on the Fahrenheit scale.

Exercise 1:

Write a program that can convert degrees Fahrenheit to degrees Celsius, or vice versa.

Exercise 2:

Improve your program so that it converts the given numerical value to both Degrees Celsius and to Degrees Fahrenheit. For example, if the user of the program types in 30, your program should say how much 30 °C is in Degrees Fahrenheit and how much 30 °F is in Degrees Celsius.

Exercise 3:

Improve your program so that if the user types in the value 0 (zero), the program prints a table which looks like the following

Celsius	Fahrenheit
-20.00	-4.00
-15.00	5.00
-10.00	14.00
-5.00	23.00
0.00	32.00
5.00	41.00
10.00	50.00
15.00	59.00
20.00	68.00
25.00	77.00

You need an if construct to test whether the given value is zero. In addition you must print the temperature table in a loop. Program `Largeint.java` is an example where if constructs are used. A while loop is demonstrated in program `Whilesum.java`. Programs `Distance.java` and `Formatting.java` show how to format the output on the screen.

Exercise 4:

Improve your program so that the above temperature table is printed with a separate static method. The method should look like

```
public static void print_temperature_table()
{
    ...
}
```

and it can be called from the main() method in the following way

```
if ( given_temperature == 0 )  
{  
    print_temperature_table() ;  
}
```

Program Letters.java is an example that demonstrates how a parameterless method can be called.

EXERCISES WITH PROGRAM `Animals.java`

Exercise 1:

Write a new method named `make_stomach_empty()` to class `Animal` in `Animals.java`. The new method could be called

```
animal_object.make_stomach_empty() ;
```

and it should make `stomach_contents` reference an empty string `""`.

Exercise 2:

Add the new data field

```
String animal_name ;
```

to class `Animal` in program `Animals.java`. You have to modify the first constructor of the class so that an `Animal` object can be created by writing

```
Animal named_cat = new Animal( "cat", "Ludwig" ) ;
```

You also need to modify the copy constructor so that it copies the new data field. Method `make_speak()` must be modified so that it prints something like

```
Hello, I am a cat called Ludwig.  
I have eaten: ...
```

Exercise 3:

Modify method `make_speak()` in program `Animals.java` so that it prints something like

```
Hello, I am ...  
My stomach is empty.
```

in the case when `stomach_contents` references just an empty string. The stomach is empty as long as method `feed()` has not been called for an `Animal` object. You can use the standard string method `length()` to check if the stomach is empty. Method `length()` can be used, for example, in the following way

```
if ( stomach_contents.length() == 0 )  
{  
    // stomach_contents references an empty string.  
    ...
```

Exercise 4:

Write a default constructor for class `Animal` in program `Animals.java`. A default constructor is such that it can be called without giving any parameters. The default constructor should initialize the data fields so that the program lines

```
Animal some_animal = new Animal();  
some_animal.make_speak();
```

would produce the following output on the screen

```
Hello, I am a default animal called no name.  
...
```

Exercise 5:

Modify program Animals.java so that you add there a new class named Zoo. You can write this new class after the Animal class. Objects of class Zoo should be objects that contain a set of Animal objects. You do not necessarily need a constructor in class Zoo if you initialize the data members when they are declared. The Zoo class should have a method named add_animal() with which a new Animal object can be added to the zoo. Moreover, the Zoo class should contain a method named make_animals_speak(). Inside this method the make_speak() method should be called for each Animal object. The Zoo class can look like the following:

```
class Zoo
{
    Animal[] animals_in_zoo = new Animal[ 20 ] ;

    int number_of_animals_in_zoo = 0 ;

    public void add_animal( Animal new_animal_to_zoo )
    {
        ...

    public void make_animals_speak()
    {
        for ( int animal_index = 0 ;
              animal_index < number_of_animals_in_zoo ;
              animal_index ++ )
```

```
{  
    ...
```

This Zoo class contains an array of type `Animal[]` which stores references to `Animal`-objects. The variable `number_of_animals_in_zoo` is used to count how many animals have been added to the zoo. By studying program `Olympics.java`, you can find out how an array of objects can be used.

You can test your new Zoo class by writing the following statements to method `main()`:

```
Zoo test_zoo = new Zoo() ;  
  
test_zoo.add_animal( cat_object ) ;  
test_zoo.add_animal( dog_object ) ;  
test_zoo.add_animal( another_cat ) ;  
test_zoo.add_animal( some_animal ) ;  
  
test_zoo.make_animals_speak() ;
```

Exercise 6:

In this exercise we will modify only the internal structure of class `Animal`. The functionality of the methods of class `Animal` may not change although their internal statements will be modified. This means that you do not need to modify the `main()` method.

Modify the data field `stomach_contents` in class `Animal` so that it becomes an array of type `String[]`. You can write it as follows

```
String[] stomach_contents = new String[ 30 ] ;
```

This array can store 30 references to `String` objects. When an `Animal` object is being fed, the given "string of food" should always be stored in the first free array position. To store the strings into this array, you need to have a variable that counts how many times the animal in question has been fed. For this purpose you can use a data field like

```
int number_of_feedings = 0 ;
```

which can also serve as an array index.

To use the above-defined array, you need to modify the internal statements of the methods in class `Animal`. For instance, the `feed()` method must be modified so that the given food is stored to the array. This can be accomplished with a statement like

```
stomach_contents[ number_of_feedings ] =  
                                food_for_this_animal ;
```

Method `make_speak()` can find out whether or not the the animal has been fed by checking the value of `number_of_feedings`. Stomach contents can be printed with a loop that begins in

the following way

```
for ( int food_index = 0 ;  
      food_index < number_of_feedings ;  
      food_index ++ )  
{  
    System.out.print( ...  
        // print one "string of food" at a time
```

Exercise 7:

Derive from class `Animal` a new class named `Carnivore`. (A carnivore is an animal that eats other animals.) The `Carnivore` class must contain a new version of method `feed()`. With this new method it will be possible to feed other animals to a `Carnivore` object. The new `feed()` method can begin in the following way:

```
public void feed( Animal animal_to_be_eaten )  
{
```

An animal can be eaten so that the data field `species_name` is copied to the stomach of a `Carnivore` object. Inside the new `feed()` method, the data field `species_name` of the given `Animal` object can be referred to in the following way:

```
animal_to_be_eaten.species_name
```

The following statement would copy a reference to this data field

```
stomach_contents[ number_of_feedings ] =  
    animal_to_be_eaten.species_name ;
```

You can build the new Carnivore class gradually so that you first write the class so that it only has a constructor. After you have found out that the new class works so that you can construct and use Carnivore objects, you can add the new feed() method.

You can write the new class after the Animal class into the existing .java file. (A single .java file may contain several classes if the classes are not declared with the keyword public.)

Please, study the programs BankPolymorphic.java and Windows.java to find out how a class can be derived from an existing class. In Java, a new class is derived with the keyword extends in the following way:

```
class Carnivore extends Animal  
{  
    ...
```

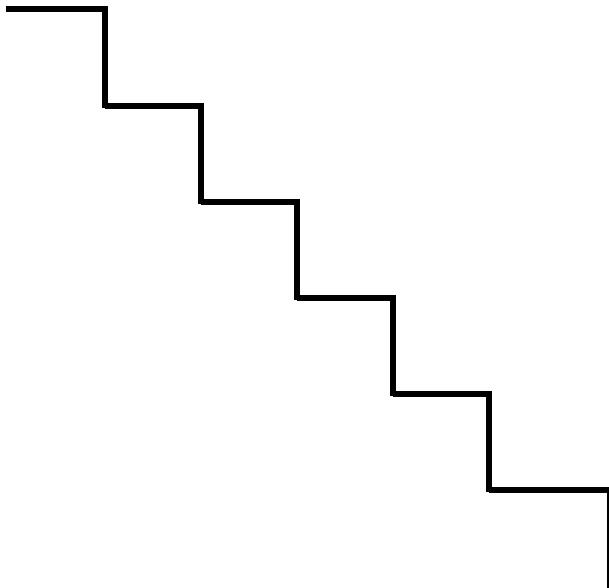
Your Carnivore class could be tested with statements like:

```
Carnivore tiger = new Carnivore( "..." ) ;  
Animal      cow  = new Animal( "..." ) ;  
  
tiger.feed( cow ) ;
```

EXERCISES WITH PROGRAM "HelloApplet.java"

Remember: when you work with applets, you need both a .java file and the corresponding .html file.

1. Study the program "HelloApplet.java", and modify the program so that it draws descending steps to the screen in the following way



You can draw these steps by calling the `drawLine()` method many times, but I recommend that you draw the steps by calling method `drawLine()` inside a loop. The structure of the needed `paint()` method can be like

```

int line_start_point_x = 10 ;
int line_start_point_y = 10 ;

while ( ...
{
    graphics.drawLine( line_start_point_x,
                       line_start_point_y,
                       ... // drawing a horizontal line

    line_start_point_x = line_start_point_x + 30 ;

    graphics.drawLine( line_start_point_x,
                       line_start_point_y,
                       ... // drawing a vertical line

    line_start_point_y = line_start_point_y + 30 ;
}

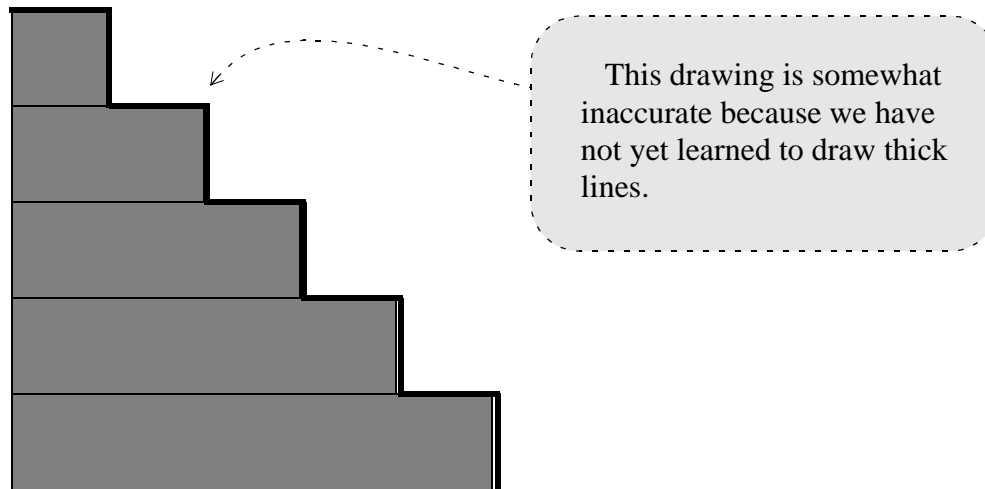
```

A variable like

```
int step_counter = 0 ;
```

could be useful in the paint() method. With this variable you can count how many steps have been drawn and terminate the loop when necessary.

2. Modify the loop that you wrote in the previous exercise so that the Graphics method `fillRect()` will be used to draw a rectangle for each step as shown in the drawing below. While drawing the rectangles you should use a color other than black. By using the `drawRect()` method you can draw an outline for each rectangle. Program `EarthAndMoonApplet.java` shows how the drawing color can be changed with the `setColor()` method. The `fillRect()` method is used, for example, in program `DrawingDemoApplet.java`.



3. Add the following data field to the applet class.

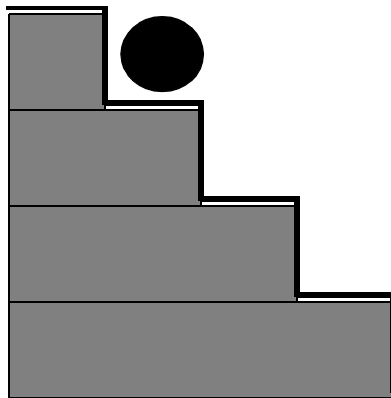
```
int painting_counter = 0 ;
```

When you add the statements

```
painting_counter ++ ;  
graphics.drawString( "Painted " + painting_counter +  
    " times.", 150, 10 ) ;
```

to the paint() method, you can find out that the value of the new data field is incremented always when you first minimize and then "de-minimize" the applet window. The paint() method is always called when the applet window is exposed.

After you have done the above-mentioned modifications, your task is to modify the program so that a ball is drawn near the first step as shown by the following illustration.



This drawing describes the situation when paint() is called for the first time. Your program may produce more steps than is shown in this drawing.

When the applet window is first minimized and then "de-minimized", the ball should move near the step below. After the ball has reached the bottom step, the next call to the paint() method should draw the ball again near the first step.

4. With the following call to the setColor() method

```
graphics.setColor(  
    new Color( line_start_point_y * 0x10000 ) ) ;
```

the rectangles that are drawn for each step are drawn with variations of the red color. Explore class Color and find out how these rectangles can be drawn with different variations of blue or green.

5. Improve the applet so that the applet first finds out what is the height of the applet area, and then the applet draws exactly 7 steps so that these steps occupy the whole height of the applet area. The diameter of the ball must be 3/4 of the height of a step. (Study the program DrawingDemoApplet.java)

NOTE: When you are testing an applet, you can add calls to the System.out.print() method to your program. These method calls print text lines to the command prompt window that is opened by the AppletViewer program. A method call like

```
System.out.print( "\n " + line_start_point_y ) ;
```

could be used to test how the value of variable line_start_point_y changes.

EXERCISES WITH PROGRAM "WormApplet.java"

Copy the files WormApplet.java and Worm.html into a local work folder on your computer.

Exercise 1

Modify the program, i.e., the run() method, so that the worm will not "jump" from right to left in the applet area on the screen. The worm should make also this traversal as worms usually do, which means that it should stretch and shrink while going to the left.

In this exercise you can copy the while loop that controls the worm movement to the right and modify it so that it makes the worm go to the left. The main while() loop of the run method could thus contain two internal while loops which in turn contain two for loops. When moving to the left, the left end of the worm should stretch first, and after this the right end of the worm should shrink.

Exercise 2

Add the following data fields to the applet class

```
int  number_of_full_traversals  =  0  ;  
int  painting_counter          =  0  ;
```

Use these variables to count how many times to worm visits the right border of the applet area, and how many times the paint() method is called. You can draw the values of these variables to the upper left corner of the applet area.

Exercise 3

Study the program `KeyboardInputDemoApplet.java` and modify `WormApplet.java` so that the speed of the worm can be adjusted with the Arrow Up and Arrow Down keys. To implement this feature, you could use a data field such as

```
int worm_slowness_factor = 50 ;
```

whose value is incremented when the Arrow Down key is pressed and decremented when the Arrow Up key is pressed. When this data field is in use, different kinds of worm speeds can be achieved with method calls such as

```
let_this_thread_sleep( worm_slowness_factor ) ;  
let_this_thread_sleep( 5 * worm_slowness_factor ) ;
```

In addition you have to modify the program in the following ways:

- The applet class must implement two interfaces in the following way:

```
implements Runnable, KeyListener
```

- The `init()` method must register the applet as a key listener:

```
public void init()  
{    ...  
    addKeyListener( this ) ;  
}
```

- You need to add the methods that react to keyboard events and you must import the classes in the `java.awt.event` package.

In this exercise you should use an if construct that prevents the value of `worm_slowness_factor` from becoming zero or negative.

When you work with an applet that should react to key pressings, it is possible that you need to click the applet window with a mouse before the program starts receiving keyboard input.

Exercise 4:

Improve the program so that the color of the worm is set to

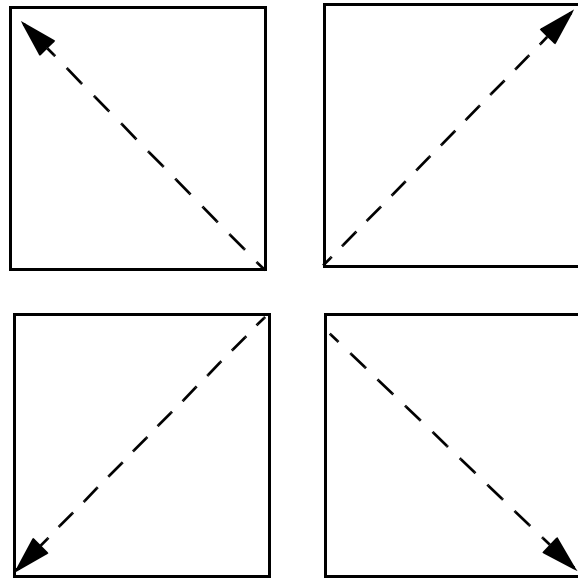
- blue when the B key is pressed,
- pink when the P key is pressed,
- magenta when the M key is pressed, and
- cyan when the C key is pressed.

Exercise 5:

The standard class `Math` provides the static method `Math.random()` that can be used to generate random numbers. Improve your program so that the worm is given a random color when the key R is pressed. The `Color` class provides constructors with which it is possible to specify a color as numbers. The `Math.random()` method is used in program `MathDemo.java` that can be found among the simple Java programs that are provided at <http://naturalprogramming.com>.

EXERCISES WITH PROGRAM "DrawingRectanglesApplet.java"

DrawingRectanglesApplet.java is a modified version of program DrawingLinesApplet.java. This program draws rectangles to the screen. The program is able to draw a rectangle "in a correct way" regardless of the direction to which the mouse moves. The possible mouse movements and resulting rectangles are shown below. Various drawing programs usually produce rectangles in this way.



The program has a method named `draw_rectangle()` that draws the rectangles. Study this method and do the following exercises.

Exercise 1

Draw a picture (a .jpg or a .gif image) inside the drawn rectangles. By studying program SinglePictureApplet.java you can find out how pictures can be drawn

You can draw the picture inside method draw_rectangle(). When the picture is drawn before the rectangle, the rectangle form a frame for the picture. You should use such a version of the drawImage() method that it draws the picture so that its size is the same as the size of the rectangle.

Exercise 2

By studying the init() method you can see that with a method call like getContentPane().setBackground(...) you can set the background color of the applet area. Add the necessary statements to the "empty" methods mouseEntered() and mouseExited() so that the background color is set to yellow when the mouse enters the applet area, and the background color changes back to white when the mouse leaves the applet area. It may be necessary to repaint the applet area in order to activate a new background color.

Exercise 3

Modify the program so that if a mouse button is pressed down and released while the Alt keyboard key is simultaneously pressed down, all rectangles and images on the screen are removed, i.e., the applet area is cleared.

- You can "remove" all drawn rectangles simply by zeroing the variable that counts how many rectangles have been drawn. A repainting operation is needed after this.
- A method named `isAltDown()` can be called for `MouseEvent` objects. This method returns true if the Alt key was pressed during the mouse event.
- The drawing of a new rectangle may not be allowed in the `mousePressed()` and `mouseDragged()` methods when the Alt key is down. Then it might be best to put the removal of the rectangles into the `mouseReleased()` method when the Alt key is pressed down. The state of the Alt key must thus be checked in all these three methods.

Exercise 4

Improve the program so that the last drawn rectangle and picture can be removed when some other mouse button than the left button is used. By studying program `MouseDemoApplet.java` you can find out how to discover which mouse button is used in a mouse operation. You can modify the mouse methods according to the same logic as was used in the preceding exercise. You do not have to use any boolean variables in this exercise.

Exercise 5

Study the program `WormOffscreenApplet.java`, and implement so called double buffering feature into the applet you have been working with. No `update()` method is needed in your applet but from the `paint()` method you must remove the call to the superclass `paint()` method.

After this feature has been implemented everything is first drawn to the offscreen image, and then this image is drawn onto the applet area. The features implemented in Exercise 2 will not work after this exercise has been implemented. The reason for this is that the offscreen image hides all the background color. Consider how to re-implement Exercise 2 so that the background color changes as specified.

EXERCISES WITH PROGRAM "MovingBallApplet.java"

Exercise 1:

Add new buttons to the applet so that the diameter of the ball can be adjusted with the new buttons. In the current version of the program the diameter is 80 points (pixels).

- You need two new JButton objects in the program. Add these new buttons first, and check that they really appear on the screen. After this you can make the buttons do something.
- You need a new data member (variable) which contains the current diameter of the ball. The variable can be declared in the following way:

```
int ball_diameter = 80 ;
```

Exercise 2:

Modify the program further so that the ball diameter (size) can be adjusted with a so-called scrollbar. A scrollbar can be created with the standard class JScrollBar. See program RectangleApplet.java to find out how a JScrollBar object can be created. Check also the Java API documentation for more information about class JScrollBar.

Exercise 3:

Modify the program so that it shows three balls on the screen at the beginning. Only one of the balls is active so that its color and diameter can be modified. Add a new button with which the active ball can be changed.

To make this modification, it is best that you use an existing class named Ball that can be found in Ball.java. You need to put the Ball.java file to the same folder (directory) where our own program is. You do not need to modify the file Ball.java. (You can also study a program named MovingBallOOApplet.java, which uses the Ball class and is an object-oriented version of the original program MovingBallApplet.java.) When the Ball class is at your disposal, you can make declarations such as:

```
Ball first_ball, second_ball, third_ball ;  
Ball active_ball ;
```

You can create Ball objects at the end of the init() method:

```
first_ball = new Ball( applet_width / 2 - 160,  
                       applet_height / 2 - 40,  
                       Color.red ) ;  
  
second_ball = new Ball( applet_width / 2 - 40,  
                        applet_height / 2 - 40,  
                        Color.green ) ;  
  
third_ball = new Ball( applet_width / 2 + 80,  
                       applet_height / 2 - 40,  
                       Color.blue ) ;  
  
active_ball = first_ball ;  
active_ball.activate_ball() ;
```

When you use a special object reference like `active_ball`, it is easy in methods like `actionPerformed()` and `paint()` to reference automatically the currently active ball. When you want to change the active ball, you can use a construct like

```
....
else if ( event.getSource() ==
           button_to_change_active_ball )
{
    active_ball.deactivate_ball() ;

    if ( active_ball == first_ball )
    {
        active_ball = second_ball ;
    }
    else if ( active_ball == second_ball )
    {
        active_ball = third_ball ;
    }
    else if ( active_ball == third_ball )
    {
        active_ball = first_ball ;
    }

    active_ball.activate_ball() ;
}
```

Exercises with program ...

To be continued.