Chapter 9 Review Exercise Solutions

R9.1

The following require a cast:

```
c = i; // c = (C) i;
i = j; // i = (I) j;
```

R9.2

None of them will throw an exception.

R9.3

The following are legal:

a. `e = sub;`
c. `sub = (Sandwich) e;`

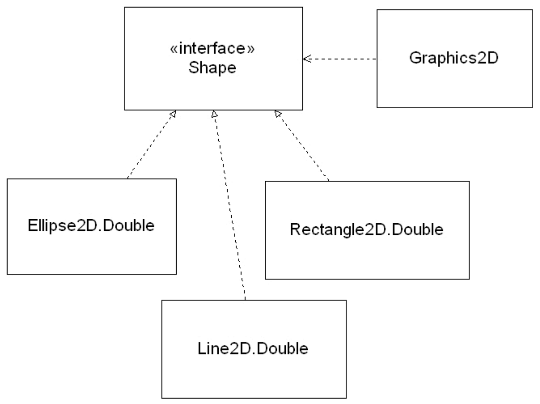
The following statement will compile but throw an exception at runtime.

f. `e = (Edible) cerealBox;`

R9.4

When casting a primitive type, the user acknowledges to the compiler that information may be lost. When casting objects, the user acknowledges to the compiler that the cast may cause an exception.

R9.5



R9.6

a. `Rectangle a = r;`
b. `Shape b = r;`
f. `Serializable f = r;`
g. `Object g = r;`

R9.7

The call `Rectangle r = s.getBounds();` is an example of polymorphism because the `Shape` interface declares the method, and each class that implements the interface (`Rectangle2D.Double`, `Ellipse2D.Double`, `Line2D.Double`, etc.) provides its own implementation of the method. The correct implementation is picked at runtime.

R9.8

Calls to constructors and calls to `static` methods are two kinds of method calls that use early binding in Java.

R9.9

In the first implementation, you need to modify `Employee` so that it implements the `Measurable` interface. In the second implementation, you need to construct a `Measurer` object to carry out the measurement, and pass that object to the `DataSet` constructor, so that it can be used to perform the measurements.

The second implementation is easier because the responsibility of measuring does not lie on the added objects. This way, you can define measurers to take on any kind of measurement, and we do not need to modify the `Employee` class to implement `Measurable`.

R9.10

In the Section 9.1 implementation, the compiler would complain that the add method cannot take `String` object when it is expecting an object of type `Measurable`.

In the second implementation in Section 9.4, the `add` method does not complain since it will take any object. But it will throw an exception when the measurer tries to convert the `String` reference to a `Rectangle` reference.

R9.11

We would need to put each class (`DataSetTester3` and `RectangleMeasurer`) in its own file. No further change is required because the `RectangleMeasurer measure` method does not access variables from the surrounding scope.

R9.12

A *callback* is a mechanism used to call back a specific method when it is needed, to provide more information. Another use for a callback for the `DataSet` class can be a `Filter` object that filters out certain information before being passed to the `DataSet` object.

R9.13

The `f` method can access the variables `b`, `t`, and `x`.

R9.14

The compiler gives an error saying that the local variable is accessed from within inner class and needs to be declared `final`:

```
local variable a is accessed from within inner class; needs to be
declared final
```

If we change the variable so that it is declared as `final`, then the inner class can access it, provided it does not try to assign a new value.

R9.15

We would need to put each class (`InvestmentViewer1` and `AddInterestListener`) in its own file. With this change, the class `AddInterestListener` is no longer able to access the `account` variable. Thus, we need to add a `"private BankAccount account"` field to the class `AddInterestListener`, and have it receive the bank account in the constructor.

R9.16

An event is an external activity to which a program may want to react. For example, “user clicked on button X” is an event.

An event source is the user interface component that generates a particular event. Event sources report on events. When an event occurs, the event source notifies all event listeners.

An event listener belongs to a class that is provided by the application programmer. Its methods describe the actions to be taken when an event occurs.

R9.17

With a console application, the programmer is in control and it can ask the user for input, in the order that is most convenient for the program.

With a graphical user interface application, the programmer lays out the various elements for user input. Then the user is in control. The user can click and type into the components in any order, and the programmer must be able to cope with that variety.

R9.18

An `ActionEvent` is generated when the user interface library has detected a particular user intent, such as clicking on a button or hitting `ENTER` in a text field.

A `MouseEvent` is generated whenever the user moves or clicks the mouse.

R9.19

An `ActionListener` is only interested in one notification: when the action happens.

A `MouseListener` has multiple methods because there are several potentially interesting mouse events, such as pressing, releasing, or clicking the mouse button.

R9.20

Yes–a class can be the event source for multiple event types. For example, a `JButton` is the event source for both mouse events and action events.

You can listen to the mouse events of a button, simply by installing a mouse listener:

```
button.addMouseListener(mouseListener);
```

You might want to do that for example to make the button glow in a different color whenever the mouse hovers over it.

R9.21

Every event carries the *source* object. You can retrieve it with the `getSource` method.

A mouse event has the following additional information:

- the x- and y- component of the mouse position
- the click count

An action event has the following additional information:

- the command string associated with the action
- the modifier keys held down during the action event

Hint: This information can be found in the API documentation.

R9.22

Event listeners often need to access instance fields from another class. An inner class can access the instance fields of the outer class object that created it. Thus, it is convenient to use inner classes when implementing event listeners.

If Java had no inner classes, we could pass the values of instance fields needed to be accessed to the event listener constructor, so that local references/copies can be stored in the event listener object.

R9.23

The `paintComponent` method is called by the GUI mechanism whenever *it* feels that a component needs to be repainted. That might happen, for example, because the user just closed another window that obscured the component.

A programmer calls the `repaint` method when the *programmer* feels that the component needs to be repainted (for example, if the state of the object has changed). That call is a request to the GUI mechanism to call `paintComponent` at an appropriate moment.

R9.24

A frame is a window to which you can add a component to be displayed on the screen. A frame can be displayed on its own, even if it is empty. However, we cannot simply add multiple components directly to a frame–they would be placed on top of each other. A panel is a container to group multiple user interface components together. A panel needs to be added to a frame to be displayed.