RICHARD ROBINSON

JAVA I EXAM PREPARATION

GUIs

Widgets

There are several classes in the `java.swing.*` toolkit, with the syntax

```
JComponent component = new Widget(String); // adds Widget
```

where `Widget` is the name of the widget. Interacting with such cases an event object to be created, handled by creating a listener for specific events. Such listener must be registered per widget, each causing different events. Event listeners must implement the corresponding interface and methods.

The `JButton, JCheckBox, JComboBox, JRadioButton` classes fire `ActionEvent` event objects, implement the `ActionListener` interface, and registered with the `addActionListener` method. For all other classes, the `MouseEvent` events are fired, the `MouseListener` interface is implemented, and uses the `addMouseListener` method. Typically, the `app` calls the `frame` which in turn calls the `panel`.

Containers

The main `JFrame` class represents a GUI window with title bar, resizable border, and border buttons. Apps extend this class to customize it using

```
public class ClassName extends JFrame implements ActionListener
```

The `JPanel` class is used to arrange widgets, and can also contain other panels and can be used to draw custom shapes. The `paintComponent` method is called to redraw elements and can be overridden to create custom appearances; it is called by `repaint()`.

Layout Managers

There are several types of layout managers, including:

- `FlowLayout` is arranged linearly and flows to next line if needed. It is based on the preferred size.
- `BorderLayout` adds components using cardinal directions and ignores preferred size.
- `GridLayout` arranged components in a grid and ignores preferred size.
- `BoxLayout` is similar to `FlowLayout` with advanced options, and is preferred size.

Definitions

Aggregation

Aggregation represents a *has a* relationship between two classes. A class is an aggregate if it has an attribute of a non-primitive type. This works as given by:

```
public class ClassName {
    private CustomType var; // attributes
    public ClassName(Type var) { this.var = var; } // constructor
}

// using the class in Main
CustomType var = new CustomType(values);
ClassName name = new Classname(vars);
```

With this code, you can call the attributes of the class via `name.var`.

Classes & Objects

Classes are used to define templates, and objects to instantiate classes. Objects are created and methods on objects are called. A template has common attributes (nouns) and behaviors (verbs). Each instance of a class has specific attribute values. An example is

```
public class Point {
    float x, y; // attributes from another class
    Point(float x, float y) { this.x = x; this.y = y } // constructor
    Type getFunc() { return z } // action returning value
    void actionName(Type var) { this.x = z; } // action modifying values
}
```

Then, in the main class, the object can be called via:

```
Point p1 = new Point(1, 2); // creating new instance of object
p1.getFunc(); // calling object without pars
p1.actionName(var1); // calling object with pars
```

References

The `this` reference replaces the generic variable in a class with a version specific to the variable that is called. It is used when the attributes share names with the constructor parameters so as to disambiguate; `this.par` references the attribute version of `par`, not the parameter version. That is,

```
float x;
ClassName(float a) { x = a; }
```

is equivalent to

```
float x;
ClassName(float x) { this.x = x; }
```

Consequently, a mutator method changes values of the attributes to the object it is referencing via the notation `var.changeAtt(par)`. As well, accessor methods use the result of a computation on its attributes, `Type Att = var.getAtt()`.