AppsProg Study 1

Info

Programming: The process of converting a specification into a program.
Pattern: A common solution to a common problem.

Array: A set of values of the same type. `Type name[] = {val, val}`

/Note: getting length uses `.length` as it is not a standard method/

Scanner: `new Scanner(System.in)`

Methods

- `.nextInt()` an integer value
- `.nextDouble()` a double value
- `.nextLine()` a String
- `.nextChar()` a char value

Words To Code

Attempting to understand the patterns used by translating steps 1:1 into functionality.
e.g.
Read in cards = `READ LOOP`
Until user breaks = `while (input != -1)`
Then show the highest card = `MAX PATTERN`

Patterns

COUNT

```
count = 0
<for each item>
  if (pass condition)
      count++
```

```
  @Override
  public double getArea() {
      return size*size
  }
}
```

Note: to implement multiple interfaces, simple seperate them via a comma

Polymorphism Allows for a single object to have many types.

Superclasses Unlike an interface, a superclass also defines the common methods, as well as being able to provide common fields and non-public members.

- When the superclass contains methods that may vary in implementation, they are declared as `abstract`.
- A subclass may not inherit more than one superclass.
- If a superclass contains abstract methods, it must be defined as `public abstract class`.
- Fields in a superclass should be `protected`.
- Subclasses `extends` their super.
- Non-abstract methods may be overridden with `@Override`
- A method defined in the superclass can be called within the subclass with `super.aMethod()`

Note: the constructor of a sub-class must first call the super-constructor

```
public class Square() extends Polygon {
  public Square(size) {
      super()
      this.size = size
  }
}
```

#Uni/AppsProg

```
(money, drugs) -> {
  //line
}
```

Event Something that happens in a GUI application. There are two types of events:

- ActionEvent
- KeyEvent

Events can be defined in a number of ways:

```
public void start(Stage stage) {
  //Inner-Class
  btn.setOnAction(new buttonHandler())

  //Anonymous Inner-Class
  btn.setOnAction(new EventHandler<ActionEvent>() {
      @Override public void handle(ActionEvent e) {
          //use
      }
  })

  //Lambda Expression
  btn.setOnAction(event -> //use)
}

private class buttonHandler implements EventHandler<ActionEvent> {
  @Override public void handle(ActionEvent e) {
      //use
  }
}
```

TextField TextFields have `.getText()` and `.setText()` methods, each returning or providing a String value.

```
Integer.parseInt(tF.getText())
```

```
import java.fxml.*

@Override public void start(Stage stage) throws Exception {
  FXMLLoader loader = new FXMLLoader(getClass.getResource
                                                    ("file.fxml"))
  Parent root = loader.load()
  Stage.setScene(new Scene(root))
  stage.show()
}
```

Stage also has method `.sizeToScene()` which condenses the window to content.

To use a dollar sign within quotes it can be escaped as so: `"\$memes"`

MVC

Model Java objects that represent the data of the application and the operations on that data.

View The components that represent the GUI of the application, observing the data Models.

Controller The components that handle user interaction. Controllers observe events occurring in the views.

Setting Up the Models

For the view to observe or see Model values, getters and setters are required.

JavaBeans Properties Any method pertaining to a single value of an object is a property. For instance, the pairing of `.getName()` and `.setName()` form the Name property.

JavaFX Properties Observable properties following the observer pattern.

- **Immutable Property** a property that never changes.

```
private final int val
```