Lab Quiz 2 Template

TA Instructions

Before Lab:

- Copy this template into a new file and put it in the Lab Quiz 2 folder with your lab section in the name. Do not edit this template directly since other TAs will need to copy this
- Fill in all the personalizations for each question template **before your lab section**
    - Feel free to alter the questions beyond the templates as long as you're testing the same concepts. The template provides a minimal amount of variance between labs
- Have a plan to display the quiz to your students
    - Setup a github repo and have students view the code on their computers with close supervision to ensure they only access the quiz code
    - -or- Notify Jesse via Slack that you want your quiz printed

During Lab:

- Give them 1 hour to complete the Quiz. You don't have to start at the beginning of lab but you must end the quiz 1 hour after you start
- **Team Adjustments:** For the remaining time, address any team reformations that may be necessary due to resignations and non-participants. Use your discretion in rearranging teams and bring up any tough calls in the Slack channel for help deciding.

Grading:

- Start grading as the quiz ends. When you're done grading, add their scores into AutoLab and release the grades for your section. Their grades should be posted within 24 hours of taking the quiz. You can hand the quizzes back next week
- Run your code to be 100% sure of the answers

Student Instructions

- This is an exam environment
    - No talking
    - No material allowed except blank paper and writing utensils
    - Only ask TA's questions clarifying what a question is asking

Q1

```
abstract class Student(var grade: Double) {

 def showGrade(): Double

 def takeTest(gradeDelta: Double): Unit = {
   this.grade += gradeDelta
 }

 def cheat(): Unit = {
   this.grade = 0.0
 }

}

class GoodStudent(constructorParam: Double) extends Student(constructorParam) {

 override def showGrade(): Double = {
   this.grade + 0.5
 }

}

class BadStudent(constructorParam: Double) extends Student(constructorParam) {

 override def showGrade(): Double = {
   this.grade - 0.5
 }

 override def takeTest(gradeDelta: Double): Unit = {
   this.grade += (gradeDelta - 0.2)
 }

}

def accumulate(students: List[Student]): Double = {
 var total = 0
 for(student <- students){
   total += student.showGrade()
 }
 total
}

def exam(students: List[Student], gradeDelta: Double): Unit = {
 for(student <- students){
   student.takeTest(gradeDelta)
 }
```

```
}
```

Q1, Part 1 - What will be printed when this main method is run?

```
def main(args: Array[String]): Unit = {
  val studentA: Student = new GoodStudent(90)
  val studentB: BadStudent = new BadStudent(70)
  val studentC: GoodStudent = studentA

  println(studentA.showGrade())
  println(studentB.showGrade())
  studentA.takeTest(5.0)
  studentB.takeTest(5.0)
  println(studentC.showGrade())
  studentC.cheat()
  println(studentA.showGrade())
  studentB.takeTest(10.0)
  println(studentB.showGrade())
}
```

Q1, Part 2 - What will be printed when this main method is run?

```
def main(args: Array[String]): Unit = {
  val studentA: Student = new GoodStudent(90)
  val studentB: BadStudent = new BadStudent(70)
  val studentC: GoodStudent = studentA
  val studentList: List[Student] = List(studentA, studentB, studentC)

  println(accumulate(studentList))
  exam(studentList, -10.0)
  println(accumulate(studentList))
}
```

Q2

```
class Model {

 var dollars: Int = 0

 def displayField(): Int = {
   this.dollars
 }

 def winMoney(value: Int): Unit = {
   this.dollars += value
 }

 def gambleMoney(): Unit = {
   // Assume this rounds down to nearest Int
   this.dollars /= 2
 }

}


class Controller(model: Model) {

 def winMoneyPressed(event: ActionEvent): Unit = model.winMoney(100)

 def gambleMoneyPressed(event: ActionEvent): Unit = model.gambleMoney()

 def userAction(event: KeyEvent): Unit = {
   event.getCode.getName match {
     case "A" => model.winMoney(100)
     case "B" => model.winMoney(200)
     case "C" => model.gambleMoney()
     case "D" => model.winMoney(300)
     case "E" => model.winMoney(400)
     case "X" => model.gambleMoney()
     case _ => model.winMoney(-100)
   }
 }

}

class QuizButton(display: String, action: EventHandler[ActionEvent]) extends Button {
 val size = 200
 minWidth = size
 minHeight = size
 onAction = action
 text = display
 style = "-fx-font: 30 ariel;"
```

```
}


object View extends JFXApp {

 val model: Model = new Model()
 val controller: Controller = new Controller(model)

 var textField: TextField = new TextField {
   editable = false
   style = "-fx-font: 26 ariel;"
   text.value = model.displayField().toString
 }

 stage = new PrimaryStage {
   title = "Quiz GUI"
   scene = new Scene() {
     content = List(
       new GridPane {
         add(textField, 0, 0, 2, 1)
         add(new QuizButton("Win Money", controller.winMoneyPressed), 0, 1)
         add(new QuizButton("Gamble Money", controller.gambleMoneyPressed), 1, 1)
       }
     )
   }

   addEventFilter(KeyEvent.KEY_PRESSED, controller.userAction)

   // update the display after every event
   addEventFilter(Event.ANY, (event: Event) => textField.text.value =
model.displayField().toString)

 }

}
```

Q2 – Take the following five sequences of inputs, and state what the state of the model field will be after the sequence. (Assume the app is resetting to its starting state between sequences)

Sequence 1

- Press ‘A’ key
- Press ‘B’ key
- Press ‘C’ key
- Press ‘D’ key
- Press ‘E’ key
- Press ‘Z’ key

Sequence 2

- Press ‘Win Money’ button
- Press ‘Win Money’ button
- Press ‘Win Money’ button
- Press ‘Gamble Money’ button

Sequence 3

- Press ‘Win Money’ button
- Press ‘Gamble Money’ button
- Press ‘Gamble Money’ button
- Press ‘A’ key
- Press ‘Z’ key

Sequence 4

- Press ‘Z’ key
- Press ‘Z’ key
- Press ‘Z’ key
- Press ‘Z’ key
- Press ‘Z’ key
- Press ‘Gamble Money’ button

Sequence 5

- Press ‘A’ key
- Press ‘Gamble Money’ button
- Press ‘B’ key
- Press ‘Win Money’ button
- Press ‘Win Money’ button
- Press ‘Win Money’ button
- Press ‘Win Money’ button
- Press ‘Gamble Money’ button