College of Engineering, Construction & Living Sciences
Bachelor of Information Technology
IN721: Mobile Application Development
Level 7, Credits 15

Practical 02: Kotlin 2

Assessment Overview

In this assessment, you will solve five coding problems using **Kotlin** in **IntelliJ IDEA**. This assessment contributes 5% towards your final mark in **IN721: Mobile Application Development**.

Learning Outcomes

At the successful completion of this course, learners will be able to:

1. Implement & publish complete, non-trivial, industry-standard mobile applications following sound architectural & code-quality standards.
2. Identify relevant use cases for a mobile computing scenario & incorporate them into an effective user experience design.
3. Follow industry standard software engineering practice in the design of mobile applications.

Assessment Table

Assessment Activity	**Weighting**	**Learning Outcomes**	**Assessment Grading Scheme**	**Completion Requirements**
Practicals	10%	2, 3	CRA	Cumulative
Project	70%	1, 2, 3	CRA	Cumulative
Presentation	20%	2, 3	CRA	Cumulative

Conditions of Assessment

You will complete this assessment during your learner managed time, however, there will be availability during the teaching sessions to discuss the requirements & your progress of this assessment. This assessment will need to be completed by **Friday, 13 August 2021**.

Pass Criteria

This assessment is criterion-referenced (CRA) with a cumulative pass mark of **50%** over all assessments in **IN721: Mobile Application Development**.

Authenticity

All parts of your submitted assessment must be completely your work & any references must be cited appropriately including, externally-sourced graphic elements. Provide your references in a **README.md** file. All media must be royalty free (or legally purchased) for educational use. Failure to do this will result in a mark of **zero** for this assessment.

Policy on Submissions, Extensions, Resubmissions & Resits

The school's process concerning submissions, extensions, resubmissions & resits complies with **Otago Polytechnic** policies. Learners can view policies on the **Otago Polytechnic** website located at https://www.op.ac.nz/about-us/governance-and-management/policies.

Submissions

You must submit all program files via **GitHub Classroom**. Here is the URL to the repository you will use for your submission – https://classroom.github.com/a/Bvbfy__J. Create a new branch called **02-kotlin-2** from the **main** branch by running the command - **git checkout -b 02-kotlin-2**. This branch will be your development branch for this assessment. Once you have completed this assessment, create a pull request & assign the **GitHub** user **grayson-orr** to a reviewer. **Do not** merge your own pull request. Late submissions will incur a **10% penalty per day**, rolling over at **5:00 PM**.

Extensions

Familiarise yourself with the assessment due date. If you need an extension, contact the course lecturer before the due date. If you require more than a week's extension, a medical certificate or support letter from your manager may be needed.

Resubmissions

Learners may be requested to resubmit an assessment following a rework of part/s of the original assessment. Resubmissions are to be completed within a negotiable short time frame & usually must be completed within the timing of the course to which the assessment relates. Resubmissions will be available to learners who have made a genuine attempt at the first assessment opportunity & achieved a **D grade (40-49%)**. The maximum grade awarded for resubmission will be **C-**.

Resits

Resits & reassessments are not applicable in **IN721: Mobile Application Development**.

Instructions - Learning Outcomes 2, 3

You have been provided a directory called **practical-02-kotlin-2** containing five **Kotlin** files. In these files, write your solutions to the five problems below.

Problem 1:

Write two classes called **SoftwareDeveloper** & **Manager** which inherit from the given **Employee** class. The **SoftwareDeveloper** class has one additional class property called **favProgLang** of type **String**. The **Manager** class also has one additional class property called **employees** of type **MutableList<Employee>** & three functions which add, remove & display all managed employees.

Use the three **SoftwareDeveloper** objects & **Manager** object in the **main** function to display the expected output.

```
  open class Employee(var id: Int, val firstName: String, val lastName: String, val salary: Int) {
      override fun toString() = "${firstName} ${lastName}"
  }

// Write your SoftwareDeveloper class here

// Write your Manager class here

  fun main() {
      val sftDevOne = SoftwareDeveloper(1, "Bert", "Watts", 100000, "Cobol")
      val sftDevTwo = SoftwareDeveloper(2, "Sara", "Cain", 75000, "Perl")
      val sftDevThree = SoftwareDeveloper(3, "Samantha", "Baker", 75000, "PHP")
      val manager = Manager(4, "Owen", "James", 150000, mutableListOf(sftDevOne, sftDevTwo))

      // Write your solution here

      // Expected output:
      // Sara Cain
      // Samantha Baker
  }
```

Problem 2:

You have been given a class called **Stack** of type **String**. Use the **Stack** object in the **main** function to display the expected output.

```
  class Stack<String>() {
      private val els = mutableListOf<String>()
      fun push(el: String) = els.add(el)
      fun peek(): String = els.last()
      fun pop(): String = els.removeAt(els.size - 1)
      fun isEmpty() = els.isEmpty()
      fun size() = els.size
      override fun toString() = "Stack[${els.joinToString()}]"
  }

  fun main() {
      val stack: Stack<String> = Stack()
      stack.push("Django")
      stack.push("Laravel")
      stack.push("Ruby on Rails")
      stack.push("Spring")

      // Write your solution here

      // Expected output:
      // Stack[Django, Laravel, Ruby on Rails]
```

```
    // Ruby on Rails is at the top of the stack
    // There are 3 item(s) in the stack
}
```

Problem 3:

You have been given a class called **Stack** of type **String**. Use the **Stack** object in the **main** function & the **readLine** function to reverse the user's input.

```
class Stack<String>() {
    private val els = mutableListOf<String>()
    fun push(el: String) = els.add(el)
    fun peek(): String = els.last()
    fun pop(): String = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun main() {
    val stack: Stack<String> = Stack()

    // Write your solution here

    // Expected output:
    // Enter some text: John Doe
    // eoD nhoJ
}
```

Problem 4:

You have been given a class called **Stack** of type **Int**. Use the **Stack** object in the **main** function & the **readLine** function to convert the user's input into binary.

```
class Stack<Int>() {
    private val els = mutableListOf<Int>()
    fun push(el: Int) = els.add(el)
    fun peek(): Int = els.last()
    fun pop(): Int = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun main() {
    val stack: Stack<Int> = Stack()

    // Write your solution here

    // Expected output:
    // Enter a number: 50
    // 110010
}
```

Problem 5:

You have been given a class called **Stack** of type **Char** & an incomplete function called **isBalanced** which accepts a **String** parameter called **sequence**. Given a **sequence** containing only parentheses, curly brackets & square brackets, determine if **sequence** is valid.

```
class Stack<Char>() {
    private val els = mutableListOf<Char>()
    fun push(el: Char) = els.add(el)
    fun peek(): Char = els.last()
    fun pop(): Char = els.removeAt(els.size - 1)
    fun isEmpty() = els.isEmpty()
    fun size() = els.size
    override fun toString() = "Stack[${els.joinToString()}]"
}

fun isBalanced(sequence: String): Boolean {
    val stack: Stack<Char> = Stack()
    val map = mapOf(
        '(' to ')', ')' to '(',
        '[' to ']', ']' to '[',
        '{' to '}', '}' to '{'
    )

    // Write your solution here
}

fun main() {
    // Expected output:
    println(isBalanced("{([])}")) // true
    println(isBalanced("{([")) // false
}
```

sequence is valid if:

- Open bracket must be closed by the same bracket type.
- Open bracket must be closed in the correct order.

```
// Example 1
Input: sequence = "()"
Output: true
// Example 2
Input: sequence = "()[]{}"
Output: true
// Example 3
Input: sequence = "{]"
Output: false
// Example 4
Input: sequence = "{[}]"
Output: false
```