AADK Task 8 Report

Title: UI Testing and Multi-Screen Art Navigation using Jetpack Compose

1. Objective

To implement UI testing in Jetpack Compose and build a multi-screen art viewer with Next and Previous navigation functionality.

2. Art Navigation Source Code

```
@Composable
fun ArtViewer() {
    var currentIndex by remember { mutableStateOf(0) }

    val artworks = listOf(
        "Starry Night",
        "Mona Lisa",
        "The Persistence of Memory",
        "The Scream"
    )

    Column(modifier = Modifier.padding(16.dp)) {

        Text(text = artworks[currentIndex])

        Spacer(modifier = Modifier.height(16.dp))

        Row {
            Button(onClick = {
                currentIndex =
                    if (currentIndex - 1 < 0) artworks.lastIndex
                    else currentIndex - 1
            }) {
                Text("Previous")
            }

            Spacer(modifier = Modifier.width(16.dp))

            Button(onClick = {
                currentIndex =
                    (currentIndex + 1) % artworks.size
            }) {
                Text("Next")
            }
        }
    }
}
```

3. UI Testing Code

```
@RunWith(AndroidJUnit4::class)
class ArtViewerTest {
    @get:Rule
```

```
    val composeTestRule = createComposeRule()

    @Test
    fun nextButton_changesArtwork() {
        composeTestRule.setContent {
            ArtViewer()
        }

        composeTestRule.onNodeWithText("Next").performClick()
        composeTestRule.onNodeWithText("Mona Lisa").assertExists()
    }
}
```

4. Explanation

- State variable currentIndex controls which artwork is displayed.
- Next button uses modulo (%) to wrap from last artwork to first.
- Previous button handles reverse wrapping logic.
- Compose automatically recomposes UI when state changes.
- UI tests simulate user clicks and verify UI output using Compose testing framework.
- Testing ensures functionality works correctly before deployment.

5. Conclusion

This task introduces professional UI testing practices and structured navigation logic. It strengthens understanding of state-driven UI updates and ensures reliable application behavior.