package com.example.pr\_13

import android.view.View

import androidx.recyclerview.widget.RecyclerView

import androidx.test.espresso.Espresso.onView

import androidx.test.espresso.action.ViewActions.\*

import androidx.test.espresso.assertion.ViewAssertions.matches

import androidx.test.espresso.matcher.BoundedMatcher

import androidx.test.espresso.matcher.ViewMatchers.\*

import androidx.test.ext.junit.runners.AndroidJUnit4

import androidx.test.filters.LargeTest

import androidx.test.platform.app.InstrumentationRegistry

import androidx.test.rule.ActivityTestRule

import org.hamcrest.Description

import org.hamcrest.Matcher

import org.junit.Assert.assertEquals

import org.junit.Rule

import org.junit.Test

import org.junit.runner.RunWith

import android.view.WindowManager.LayoutParams.TYPE\_APPLICATION\_OVERLAY

import android.view.WindowManager.LayoutParams.TYPE\_TOAST

import androidx.test.espresso.Root

import com.example.pr\_13.ToastMatcher.Companion.onToast

import kotlinx.coroutines.DelicateCoroutinesApi

import org.hamcrest.TypeSafeMatcher

/\*\*

\* Instrumented test, which will execute on an Android device.

\*

\* See [testing documentation](http://d.android.com/tools/testing).

\*/

@RunWith(AndroidJUnit4::class)

@LargeTest

class ExampleInstrumentedTest {

@Test

fun useAppContext() {

// Context of the app under test.

val appContext = InstrumentationRegistry.getInstrumentation().targetContext

assertEquals("com.example.pr\_13", appContext.packageName)

}

@DelicateCoroutinesApi

@get:Rule

@Suppress("DEPRECATION")

val ar: ActivityTestRule<MainActivity> = ActivityTestRule(MainActivity::class.java)

//val activityScenarioRule: ActivityScenarioRule<MainActivity> = ActivityScenarioRule(MainActivity::class.java)

@Test

fun runDataRowTest() {

basicInput("2021", "US")

[\\Appeal](file:///\\Appeal) - Holiday

val someAppeal = Appeals("Hello", “!”)

onView(withId(R.id.appealsList))

.check(matches(testSetUp(someAppeal)))

}

@Test

fun runEmptyFirstField () { Так?

basicInput("", “!”)

onToast(R.string.EmptyFirstFieldTest) [\\EmptyYear](file:///\\EmptyYear) = EmptyFirstField

.inRoot(ToastMatcher.isToast())

.check(matches(isDisplayed()))

}

@Test

fun runEmptySecondFieldTest() {

basicInput("Hello", "")

onToast(R.string.EmptySecondFieldTest)

.inRoot(ToastMatcher.isToast())

.check(matches(isDisplayed()))

}

@Test

fun runIncorrectFirstFieldTest() {

basicInput("!", "!")

onToast(R.string.IncorrectIncorrectFirstField)

.inRoot(ToastMatcher.isToast())

.check(matches(isDisplayed()))

}

@Test

fun runIncorrectSecondFieldTest () {

basicInput("Hello", "Hello")

onToast(R.string. SecondField)

.inRoot(ToastMatcher.isToast())

.check(matches(isDisplayed()))

}

}

private fun testSetUp(appeal:Appeals): Matcher<View> {

return object: BoundedMatcher<View, RecyclerView>(RecyclerView::class.java) {

override fun describeTo(description: Description?) {

description?.appendText("${appeal.appialLocalString}, ${appial.appealLocalSymbol}")

}

override fun matchesSafely(recyclerView: RecyclerView?): Boolean {

val list = (recyclerView?.adapter as AppealsAdapter).getResult()

return list.any { it.appealLocalString == appeal.appealLocalString && it.appealLocalSymbol == appeal.appealLocalSymbol }

}

}

}

class ToastMatcher(private val maxFailures: Int = DEFAULT\_MAX\_FAILURES) : TypeSafeMatcher<Root>() {

/\*\* Restrict number of false results from matchesSafely to avoid endless loop \*/

private var failures = 0

override fun describeTo(description: Description) {

description.appendText("is toast")

}

public override fun matchesSafely(root: Root): Boolean {

val type = root.windowLayoutParams.get().type

@Suppress("DEPRECATION") // TYPE\_TOAST is deprecated in favor of TYPE\_APPLICATION\_OVERLAY

if (type == TYPE\_TOAST || type == TYPE\_APPLICATION\_OVERLAY) {

val windowToken = root.decorView.windowToken

val appToken = root.decorView.applicationWindowToken

if (windowToken === appToken) {

// windowToken == appToken means this window isn't contained by any other windows.

// if it was a window for an activity, it would have TYPE\_BASE\_APPLICATION.

return true

}

}

// Method is called again if false is returned which is useful because a toast may take some time to pop up. But for

// obvious reasons an infinite wait isn't of help. So false is only returned as often as maxFailures specifies.

return (++failures >= maxFailures)

}

companion object {

/\*\* Default for maximum number of retries to wait for the toast to pop up \*/

private const val DEFAULT\_MAX\_FAILURES = 5

fun onToast(text: String, maxRetries: Int = DEFAULT\_MAX\_FAILURES) = onView(withText(text)).inRoot(isToast(maxRetries))!!

fun onToast(textId: Int, maxRetries: Int = DEFAULT\_MAX\_FAILURES) = onView(withText(textId)).inRoot(isToast(maxRetries))!!

fun isToast(maxRetries: Int = DEFAULT\_MAX\_FAILURES): Matcher<Root> {

return ToastMatcher(maxRetries)

}

}

}

fun basicInput(year: String, countryCode: String) {

onView(withId(R.id.editFirsField)).perform(typeText(firstfield), closeSoftKeyboard())

onView(withId(R.id.editSecondFiled)).perform(typeText(secondfield), closeSoftKeyboard())

onView(withId(R.id.displayInfo)).perform(click())

Thread.sleep(1000)

}

[\\editYear](file:///\\editYear) – editFirstField

[\\editContry](file:///\\editContry) - editSecondField