Google

ASSOCIATE-ANDROID-DEVELOPER Exam

Google Developers Certification - Associate Android Developer

Topic 1, Kotlin Exam

Question: 1

What is a correct part of an Implicit Intent for sharing data implementation?

A. val sendIntent = Intent(this, UploadService::class.java).apply { putExtra(Intent.EXTRA_TEXT, textMessage)
...
B. val sendIntent = Intent().apply { type = Intent.ACTION_SEND;
...
C. val sendIntent = Intent(this, UploadService::class.java).apply { data = Uri.parse(fileUrl)
...
D. val sendIntent = Intent().apply { action = Intent.ACTION_SEND
...

Answer: D

Explanation:
Create the text message with a string
val sendIntent = Intent().apply { action = Intent.ACTION_SEND
putExtra(Intent.EXTRA_TEXT, textMessage) type = "text/plain"
}
Reference:
https://developer.android.com/guide/components/fundamentals

Question: 2

By default, the notification's text content is truncated to fit one line. If you want your notification to be longer, for example, to create a larger text area, you can do it in this way:

A. var builder = NotificationCompat.Builder(this, CHANNEL_ID)
.setContentText("Much longer text that cannot fit one line...")
.setStyle(NotificationCompat.BigTextStyle()
.bigText("Much longer text that cannot fit one line..."))
...

B. var builder = NotificationCompat.Builder(this, CHANNEL_ID)
.setContentText("Much longer text that cannot fit one line...")
.setLongText("Much longer text that cannot fit one line..."))
...
C. var builder = NotificationCompat.Builder(this, CHANNEL_ID)
.setContentText("Much longer text that cannot fit one line...")
.setTheme(android.R.style.Theme_LongText);
...

Answer: A

Reference:
https://developer.android.com/training/notify-user/build-notification

Question: 3

Select correct demonstration of WorkRequest cancellation.

A. workManager.enqueue(OneTimeWorkRequest.Builder(FooWorker::class.java).build())
B. val request: WorkRequest = OneTimeWorkRequest.Builder (FooWorker::class.java).build()
workManager.enqueue(request)
val status = workManager.getWorkInfoByIdLiveData(request.id) status.observe(...)
C. val request: WorkRequest = OneTimeWorkRequest.Builder (FooWorker::class.java).build()
workManager.enqueue(request) workManager.cancelWorkById(request.id)
D. val request1: WorkRequest = OneTimeWorkRequest.Builder (FooWorker::class.java).build()
val request2: WorkRequest = OneTimeWorkRequest.Builder (BarWorker::class.java).build()
val request3: WorkRequest = OneTimeWorkRequest.Builder (BazWorker::class.java).build()
workManager.beginWith(request1, request2).then(request3).enqueue()
E. val request: WorkRequest = OneTimeWorkRequest.Builder (FooWorker::class.java).build()
workManager.enqueue(request) workManager.cancelWork(request)

Answer: C

Explanation: Videos:
Working with WorkManager, from the 2018 Android Dev Summit WorkManager: Beyond the basics, from the 2019 Android Dev Summit
Reference:
https://developer.android.com/reference/androidx/work/WorkManager?hl=en

Question: 4

In general, you should send an AccessibilityEvent whenever the content of your custom view changes. For example, if you are implementing a custom slider bar that allows a user to select a numeric value by pressing the left or right arrows, your custom view should emit an event of type TYPE_VIEW_TEXT_CHANGED whenever the slider value changes. Which one of the following sample codes demonstrates the use of the sendAccessibilityEvent() method to report this event.

A. override fun dispatchPopulateAccessibilityEvent(event: AccessibilityEvent): Boolean {
return super.dispatchPopulateAccessibilityEvent(event).let { completed -> if (text?.isNotEmpty() == true) {
event.text.add(text) true
} else {
completed
}
}
}
B. override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean { return when(keyCode) {
KeyEvent.KEYCODE_DPAD_LEFT -> {
currentValue--
sendAccessibilityEvent(AccessibilityEvent.TYPE_VIEW_TEXT_CHANGED)
true
}
...
}
}
C. override fun onKeyUp(keyCode: Int, event: KeyEvent): Boolean { return when(keyCode) {
KeyEvent.KEYCODE_ENTER -> {
currentValue--
sendAccessibilityEvent
(AccessibilityEvent.TYPE_VIEW_CONTEXT_CLICKED)
true
}
...
}
}

Answer: B

Reference: https://developer.android.com/guide/topics/ui/accessibility/custom-views

Question: 5

The easiest way of adding menu items (to specify the options menu for an activity) is inflating an XML file into the Menu via MenuInflater. With menu_main.xml we can do it in this way:

A. override fun onCreateOptionsMenu(menu: Menu): Boolean { menuInflater.inflate(R.menu.menu_main, menu)
return true
}
B. override fun onOptionsItemSelected(item: MenuItem): Boolean { menuInflater.inflate(R.menu.menu_main, menu) return super.onOptionsItemSelected(item)
}
C. override fun onCreate(savedInstanceState: Bundle?) { super.onCreate(savedInstanceState) setContentView(R.menu.menu_main)
}

Answer: A

Reference:
https://developer.android.com/guide/topics/ui/accessibility/custom-views