# 1. What are Listeners?

It is an interface in the View class that contains a single call-back method. These methods will be called by the Android framework when the View which is registered with the listener is triggered by user interaction with the item in UI.

call-back methods which included in event listener interface

- onClick()

  This method is called when the user touches or focus on the item using navigation-keys or trackball or presses on "enter" key or presses down on the trackball.

- onLongClick()

  This method is called when the user touches and holds the item or focuses on the item using navigation-keys or trackball and presses and holds "enter" key or presses and holds down on the trackball (for one second).

- onFocusChange()

  This method is called when the user navigates onto or away from the item, using the navigation-keys or trackball.

- onKey()

  This method is called when the user is focused on the item and presses or releases a hardware key on the device.

- onTouch()

  This method is called when the user performs a touch event, including a press, a release, or any movement gesture on the screen.

- onCreateContextMenu()

  This method is called when a Context Menu is being built.

# 2. How does Java garbage collection work?

Java garbage collection is the process by which Java programs perform automatic memory management. Java programs compile to bytecode that can be run on a Java Virtual Machine, or JVM for short. When Java programs run on the JVM, objects are created on the heap, which is a portion of memory dedicated to the program. Eventually, some objects will no longer be needed. The garbage collector finds these unused objects and deletes them to free up memory

How it works:

1. Unreferenced objects are identified and marked as ready for garbage collection
2. Marked objects are deleted
3. (Optionally) memory can be compacted after the garbage collector deletes objects, so the remaining objects are in a contiguous block at the start of the heap

# 3. What is the android manifest used for?

- To provide essential information about your app to the Android system, which the system must have before it can run any of the app's code.
- To name the Java package for the application. The package name serves as a unique identifier for the application.
- To describe the components of the application, which includes the activities, services, broadcast receivers, and content providers that compose the application. It also names the classes that implement each of the components and publishes their capabilities, such as the Intent messages that they can handle. These declarations inform the Android system of the components and the conditions in which they can be launched.
- To determine the processes that host the application components.
- To declare the permissions that the application must have in order to access protected parts of the API and interact with other applications. It also declares the permissions that others are required to have in order to interact with the application's components.

- To list the Instrumentation classes that provide profiling and other information as the application runs. These declarations are present in the manifest only while the application is being developed and are removed before the application is published.
- To make key information about the app available without having to read other files in the package or run the application.

# 4. Define the difference in Runtime and Compile Time.

Compile-time is the instance where the code you entered is converted to executable while Run-time is the instance where the executable is running.

| | |
|---|---|
| We know nothing about the program's invariants, they are whatever the programmer put in. Run-time invariants are rarely enforced by the compiler alone; it needs help from the programmer. | The program need not satisfy any invariants. In fact, it needn't be a well-formed program at all. You could feed this HTML to the compiler and watch it barf. |
| What can go wrong are run-time errors:<br>● Division by zero<br>● Dereferencing a null pointer<br>● Running out of memory<br>Also there can be errors that are detected by the program itself:<br>● Trying to open a file that isn't there<br>● Trying to find a web page and discovering that an alleged URL is not well formed | What can go wrong at compile time:<br>● Syntax errors<br>● Type Checking errors<br>● (Rarely) compiler crashes |
| If run-time succeeds, the program finishes (or keeps going) without crashing. | If the compiler succeeds, what do we know?<br>● The program was well formed---a meaningful program in whatever language.<br>● It's possible to start running the program. (The program might fail immediately, but at least we can try.) |
| Inputs and outputs are entirely up to the programmer. Files, windows on the | What are the inputs and outputs?<br>● Input was the program being |

| | |
|---|---|
| screen, network packets, jobs sent to the printer, you name it. If the program launches missiles, that's an output, and it happens only at run time | compiled, plus any header files, interfaces, libraries, or other voodoo that it needed to import in order to get compiled.<br>● Output is hopefully assembly code or relocatable object code or even an executable program. Or if something goes wrong, output is a bunch of error messages. |

# 5. What is reflection in JAVA?

It is an API with the ability to inspect and dynamically call classes, methods, attributes, etc. at runtime.

Reflection can be used to get information about

- Class The getClass() method is used to get the name of the class to which an object belongs.
- Constructors The getConstructors() method is used to get the public constructors of the class to which an object belongs.
- Methods The getMethods() method is used to get the public methods of the class to which an object's belongs.

# 6. How does gradle work behind the scene.

Gradle work with Groovy closure, for this reason look different sometimes, but the gradle call methods are like in the other classes but just in a different way. Sometimes we can't understand this is the call of the methods because we don't see the methods, and this happened because the method (from Gradle) is available in the Project class which is compiled and put in the class path of the build script. So, all that's left for you to see is only the dependencies method call, which seems kind of magical if you don't know how it's implemented. In Groovy (which Gradle builds on top), the parenthesis are optional if you have at least one argument (if there's no argument you MUST use them), for that usually we found lines like this one

```
dependencies2 {
    testCompile 'junit:junit:4.11'
}
```