Jake Taggart
Week 2 Day 1 Research

Week 2 Day 1 Research

1. What is task affinity?

A: Task affinity is the affinity or default/explicitly defined tendency for an activity to gravitate to a specific package based on the task at hand. By default, task affinity is designed to gravitate toward the package the activity at hand it is in. Thus in the default affinity, generally one application will have multiple activities that carry out a specific task. An important thing to note is that a task is a collection of activities usually related to one another, but not always. One example of a task would be looking at the application you have open on your phone and seeing what the group of activities included is generally used for to accomplish. Usually these activities in a single task are related, but sometimes they are not. For example, perhaps you have a web browser up to both check your email and to download music at the same time. While this is one application (one package), there can be various tasks at hand that use a different collection of activities to achieve each task. This often becomes a little confusing carrying out multiple tasks via multiple groups of activities in one application.

This is why launch modes and task affinity definitions are so crucial to developing an application in the most user-friendly way possible. Task affinity and launch modes are set to default in the manifest, but can be changed for very useful purposes depending on the scenario. Considering multiple apps can be up at once, it's wise to try to keep a number of instances of activities to a minimal per task. Consider that in the default launch mode, when one instance of an activity in an application is opened any prior activity in the stack remains as an instance. This is just not how things work for more anything beyond simple tasks. Thus, there are various launch modes to mitigate this (standard/default, single top, single task, and single instance) but it is crucial to remember to alter the task affinity (which package it by definition an activity will gravitate toward) when changing launch modes otherwise the launch mode change may indeed do nothing or at the very least not do what the developer wants. This is why the task affinity is so important to define when changing from default.

2. How does serialization work?

Serialization is a way of changing object's state/data into a byte stream so that it can be sent across a network infrastructure or to a hard disk/drive. Because we are sending through hardware-based components, we cannot expect these components to understand exactly what a

java object is as all the components understand is bytecode. Thus, this lets the object's state get passed through a network or a local disk without having to go to a level of coding below the

3. How do parcels work?

A: Parceling, like serialization, is a way of sending object state data to the hardware components or across the network infrastructure by changing object state data into a byte stream to allow for the hardware components to read an object's state and other necessary data. The biggest difference between serialization and parceling is that serialization is native to Java and involves reflection to infer which object states and data need to be converted to a byte stream while parcelable relies on the developer to explicitly define which objects and data need to be converted. This means that serialization results in much more garbage collection and therefore takes a longer time to carry out while parcelable is efficient with garbage buildup and is quicker because it does not use reflection.

4. What is the difference in an implicit intent and an explicit intent?

A: Explicit intent is utilized when a user is in one activity and for whatever reason (button pushed, etc.) the next activity brought up will be defined by the developer. Thus, the developer is in charge of telling exactly what activity doing a certain action in another activity will result in. Implicit intent is when the developer does not explicitly define the intent of an activity upon a certain action. Instead, implicit intent allows the OS to find an activity in a package that matches the desired task and sends them there. This can also bring up a list of choices. For example, let's say that you get an email without yet having set a default application for reading said email. In this scenario, when you click on the notification for the email, a button may pop up and ask which application you would use to check the email (web browser, built in-email app, downloaded email app, etc.). As you can see, the three applications in the parentheses can all carry out that specific activity, but because the same activity is in all three applications, the system throws the user a choice of the three.