Answers to Guide 5

1. Software Design:
   1. The important of design:
      1. Software design connects the program and user together. It tells the program what the user wants, and tells the user what the program wants.
   2. The process of defining software methods, functions, objects, and the overall structure and interaction of your code so that the resulting functionality will satisfy user requirements.
   3. Grady Booch’s four basic principles of object oriented software design
      1. Abstraction: to focus on the essential features of an element or object in OOP, ignoring extraneous or accidental properties. “Denotes the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries, relative to the perspective of the viewer”
      2. Encapsulation: the process of binding both attributes and methods together within a class. Internal details of a class hidden from outside.
      3. Modularization: the process of decomposing a problem or program into a set of modules so as to reduce the overall complexity of the problem. “Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules”.
      4. Hierarchy: “The ranking or ordering of abstraction”. System composed of interrelated subsystems, each with their own subsystems and so on until the smallest level of components are reached.
   4. Information Hiding:
      1. Basically a part of the encapsulation process within OOP.
2. Software Design patterns:
   1. Design Pattern: A general, reusable solution to a commonly occurring problem within a given context in software design. A description or template of how to solve a problem that can be used in many different situations.
   2. Patterns originated as an architectural concept by Christopher Alexander. Kent Beck and Ward Cunningham experimented with the idea of applying patterns (pattern languages) to programming.
   3. Common design patterns:
      1. Factory method: Define an interface for creating a single object, but let subclasses decide which class to instantiate. Allows a class defer instantiation to subclasses.
      2. Singleton: Ensure a class has only one instance, and provide a global point of access to it.
      3. Adapter: Convert the interface of a class into another interface clients expect. Permits classes work together that could not otherwise because of incompatible interfaces.
      4. Façade: Provide a unified interface to a set of interfaces in a subsystem. Defines a higher-level interface that makes the subsystem easier to use.
      5. Iterator: Provide a way to access the elements of an aggregate object sequentially without exposing its underlying representation.
      6. Publish/Subscribe: Define a one-to-many dependency between objects where a state change in one object results in all its dependents being notified and updated automatically.
3. No Silver Bullet
   1. There is no analog to Moore’s Law: “We cannot expect ever to see twofold gains every two years”
   2. Essential Issues: the difficulties inherent in the nature of software.
      1. Complexity: many factors – communication with team members, understanding all states of the program, complexity of function and structure, etc.
      2. Conformity: Complexity caused when conforming to other interfaces. Arbitrary complexity to interfaces.
      3. Changeability: constant due to pressures for extended function. Software is embed in cultural matrix of applications, users, laws, and machine vehicles that continually change.
      4. Invisibility: Lack of a geometric abstraction for software.
   3. Accidental Issues: the difficulties that today attend its production but are not inherent.
      1. High-level languages: Reduces accidental complexity by abstraction.
      2. Time-sharing: Preserves immediacy and allows one to maintain an overview of complexity.
      3. Unified programming environments: removes difficulty resulting from using individual programs together by providing integrated library, unified file formats, and pipes/filters.
   4. Approaches to software development:
      1. High-level languages (Ada included): Contribution from having to train in modern software design techniques for those switch to them.
      2. Object-oriented programming: abstract data types and hierarchical types.
      3. Artificial intelligence: Marginal gains.
      4. Expert systems: puts at the service of inexperience programmers the experience of the best programmers.
      5. Automatic programming:
      6. Graphical programming:
      7. Buy versus Build: The development of the mass market is the more profound long-run trend in software engineering.
      8. “I believe the single most powerful software-productivity strategy for many organizations today is to equip the computer-naïve intellectual workers who are on the firing line with personal computers and good generalized writing, drawing, file, and spreadsheet programs and then to turn them loose.”
      9. Requirements refinement and rapid prototyping: very important.
      10. “The development of approaches and tools for rapid prototyping of systems as prototyping is part of the iterative specification of requirements”
      11. Incremental development: grow, don’t build, software.
      12. Top-down design.
      13. Great designers: good design by good practices by good people.
      14. Develop ways to grow great designers.
   5. Approaches seem more similar to AGILE methodologies.
   6. Summary of Brook’s thesis: Design is the forefront of solving the inherent problems in software engineering. All other aspects merely solve the accidental problems.
   7. I neither agree nor disagree with him as I don’t know enough about software engineering.
4. Design Modeling – UML Diagrams
   1. Basic building blocks of a class diagram
      1. Nodes represent a class and contains:
         1. Top compartment: name of the class
         2. Middle compartment: attributes of the class
         3. Bottom compartment: operations the class can execute
      2. Arcs represent the parent-child and sibling relationships between classes
   2. Representation of inter-class relationships: is-a and has-a
      1. Inheritance relationships are is-a relationships.
         1. Subclass is a specialized form of the super class and the superclass is a generalization of the subclass.
      2. Association is a general has-a relationship
         1. Can link any number of classes together.
         2. Bi-directional, uni-directional, aggregation, and reflexive
         3. Represents the static relationship shared among the objects of two classes.
   3. It should be possible to automatically translate a class diagram into a program written in an object-oriented programming language. Especially with inheritance, it is easy to start with the base super-class and slowly work from general to specific, adding attributes specific to each subclass that automatically inherit the same attributes belonging to their parent classes. With associations, you can simply add those attributes to whichever classes are indicated as belonging to that link.
5. AsyncTask and AsyncTaskLoader
   1. Main thread: aka UI thread. Created when app starts and dispatches events to the appropriate UI interface elements (widgets). Also, where app interacts with components from the Android UI toolkit (android.widget and android.view packages)
   2. AsyncTask: Use to implement an asynchronous, time-consuming task on a worker thread. Performs background operations and publish results on the UI thread without manipulating threads or handlers.
   3. Purpose of each AsyncTask handler functions:
      1. onPreExecute(): invoked in UI thread before task is executed. Used to set-up the task such as displaying a progress bar.
      2. doInBackground(Params): invoked in background thread after onPreExecute(). Performs a background computation, returns the result, and passes result to onPostExecute(). Can call the publishProgress(Progress) method to publish progress.
      3. onProgressUpdate(Progress): runs in the UI thread after publishProgress(Progress) is invoked. Reports any form of progress to the UI thread while background computation is running.
      4. onPostExecute(Result): runs on the UI thread after computation is finished.
   4. Compare/Contrast AsyncTask vs. Loader:
      1. Loader: Uses LoaderManager class. AsyncTaskLoader is only a part of all the Loader subclasses.
      2. Provides a method loadInBackground() that runs in a separate thread.
         1. Results are automatically delivered to UI thread via onLoadFinsihed() callback
      3. Results will survive device-configuration changes unlike with AsyncTask
      4. Loader is destroyed when activity is destroyed unlike with Asynctask where it could linger and consume system resources.
      5. Can monitor data sources for changes and reload data upon change.
6. Connect to the Internet
   1. Best practices for network operations:
      1. Use appropriate protocols for sensitive data, such as HttpsURLConnection.
      2. Use HTTPS over HTTP and consider SSLSocketClass for authenticated, encrypted socket-level communication.
      3. Don’t use localhost network ports to handle IPC as other apps can access these. Consider Service class, etc.
      4. Validate input entered using a WebView and responses to intents that are issued against HTTP.
      5. Always perform network operations on a worker thread.
      6. Don’t trust data form insecure network protocols.
   2. JSON and Android support:
      1. Javascript Object Notation: lightweight data-interchange format.
      2. Based on subset of JavaScript Programming Language.
      3. Text format that is language independent and uses conventions of the C-family of languages.
      4. Built on 2 structures: collection of name-value pairs and an ordered list of values.
      5. Android can parse the results of web API queries that are often in JSON. Use the JSONObject and JSONArray classes.
   3. App and state of network connectivity:
      1. Monitor state of network so device doesn’t attempt calls when unavailable because network calls can be expensive and slow.
      2. May want to know what kind of connectivity because Wi-Fi networks are cheap compared to metered data networks.
7. Broadcast Receivers
   1. Android Broadcast implements the “Façade” design pattern.
   2. Broadcast Receivers listen for Broadcast intents.
   3. Broadcast intents are background operations.
      1. Two types: system broadcast intents and custom user broadcast intents.
   4. Can register broadcast receiver statically in AndroidManifest.xml or dynamically in an activity that is tied to its lifecycle.
   5. Can use LoadBroadCastManager to automatically manage security aspects.
   6. Use intent filters to specify what broadcasts to listen for.