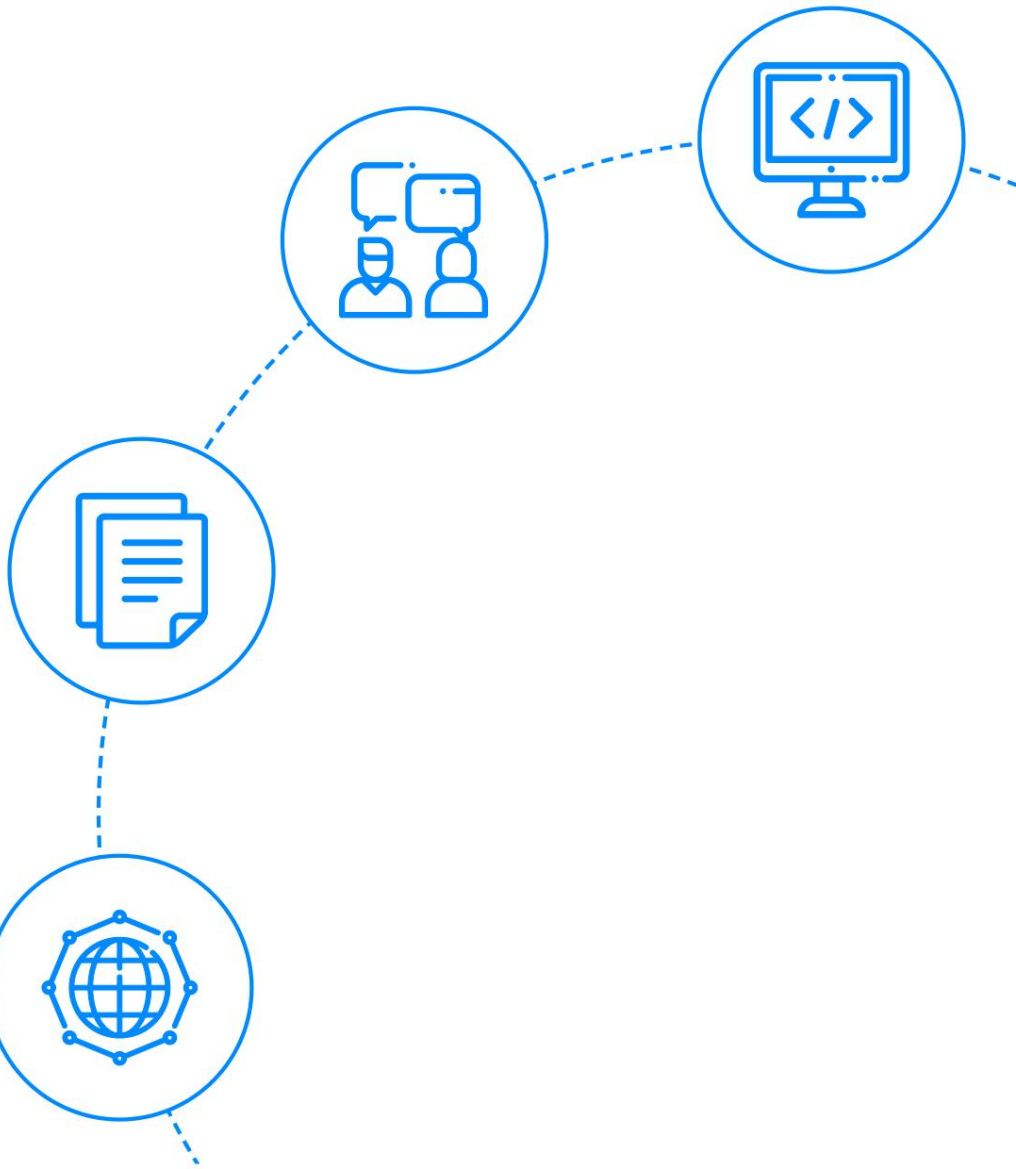
InterviewBit

निम्न स्तरीय डिजाइन साक्षात्कार प्रश्न



का लाइव संस्करण देखने के लिए पेज, यहां क्लिक करें।

अंतर्वस्तु

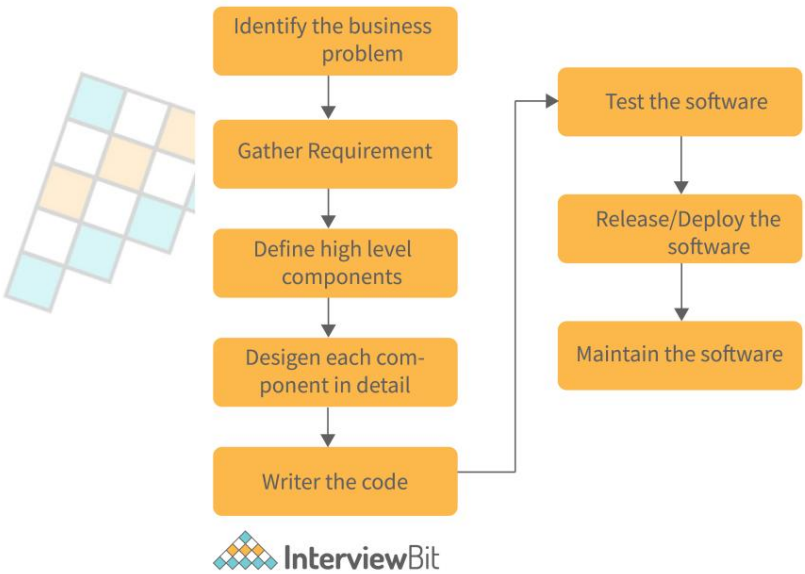
निम्न स्तरीय डिजाइन साक्षात्कार प्रश्न: फ्रेशर्स और अनुभव

लो लेवल डिज़ाइन (LLD) को क्रैक करने के टिप्स साक्षात्कार

आएँ शुरू करें

परिचय

आम तौर पर, हम किसी विशेष व्यावसायिक समस्या को हल करने के लिए सॉफ़्टवेयर विकसित करते हैं। किसी भी सॉफ़्टवेयर को विकसित करने के लिए व्यावसायिक समस्या की पहचान, पहचान की गई समस्या के लिए कार्यात्मक आवश्यकताओं को एकत्रित करना, घटकों नामक बिल्डिंग ब्लॉक्स को परिभाषित करके, व्यक्तिगत घटकों को डिजाइन करना, वास्तव में कोड लिखना, परीक्षण करना सोवेयर, सोवेयर को तैनात/रिलीज करें और सोवेयर को बनाए रखें।



वास्तविक कोड लिखने से पहले, हम 2 महत्वपूर्ण चरण करते हैं। एक उच्च-स्तरीय घटकों को परिभाषित कर रहा है जिसे आमतौर पर उच्च-स्तरीय डिज़ाइन (HLD) कहा जाता है और दूसरा प्रत्येक घटक को विस्तार से डिज़ाइन कर रहा है जिसे आमतौर पर निम्न-स्तरीय डिज़ाइन (LLD) कहा जाता है।

हाई-लेवल डिज़ाइन (HLD) क्या है?

एचएलडी में, सिस्टम के उच्च-स्तरीय आर्किटेक्चर को डिजाइन करने, उच्च-स्तरीय घटकों को उनके इंटरैक्शन के साथ परिभाषित करने और डेटाबेस डिज़ाइन पर अधिक ध्यान केंद्रित किया जाता है। HLD व्यावसायिक आवश्यकताओं को उच्च-स्तरीय समाधान में परिवर्तित करता है।

निम्न-स्तरीय डिज़ाइन (HLD) क्या है?

एलएलडी में, प्रत्येक घटक को विस्तार से डिजाइन करने पर अधिक ध्यान दिया जाता है जैसे कि किन वर्गों की आवश्यकता है, किस अमूर्त का उपयोग करना है, वस्तु निर्माण कैसे होना चाहिए, विभिन्न वस्तुओं के बीच डेटा कैसे प्रवाहित होता है, आदि। एलएलडी उच्च-स्तरीय डिजाइन को विस्तृत डिजाइन में परिवर्तित करता है। (कोड के लिए तैयार) घटक।

तकनीकी साक्षात्कारों में आजकल लो-लेवल डिज़ाइन राउंड (या) पेयर प्रोग्रामिंग राउंड (या) मशीन कोडिंग राउंड होना आम बात है ।

इस लेख में, हम चर्चा करते हैं कि एलएलडी क्यों महत्वपूर्ण है, कुछ उदाहरणों के साथ फ्रेशर्स और अनुभवी उम्मीदवारों के लिए निम्न-स्तरीय डिज़ाइन साक्षात्कार प्रश्नों की तैयारी कैसे करें , और अंत में कुछ युक्तियों के साथ समाप्त करें।

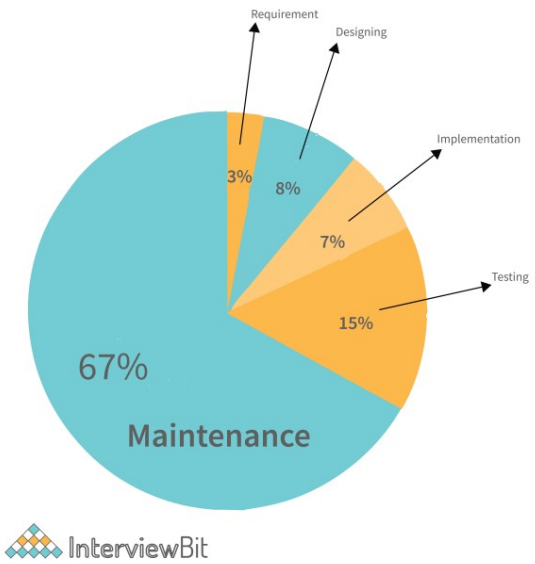
निम्न स्तरीय डिजाइन साक्षात्कार प्रश्न: फ्रेशर्स और अनुभव

1. एलएलडी क्यों महत्वपूर्ण है?

As a software developer in any company, the objective is to build software for the business of the company. Building software is a complex task. Change is the only constant thing in the world. As the software progresses over the years, it needs to incorporate a lot of changes as the requirements keep on changing. We can choose to build software in many ways but we need to build it in such a way that the software is maintainable and extensible over the years easily.

Most of the software developer interviews have Data Structures & Algorithms round(s). But it is not often for software developers to use these data structures and algorithms in the real world while building software. (If you are an experienced candidate, can you recall when was the last time you used the Dijkstra algorithm? If you are fresher, ask any of your senior colleagues the same question.)

On the other hand, Low-Level Design is a must for building a piece of software. As LLD focuses on how to build software from a set of requirements, how different components interact with each other, what responsibilities each component has etc, it is a vital exercise to be done before actually writing the code. Writing code is the easiest part of building software if we have the designs already in place. Maintaining and Extending software will be easy if designs are well thought of.



जैसा कि हम ऊपर की तस्वीर से देख सकते हैं, रखरखाव विकास जीवन चक्र में लगभग 70% प्रयास करता है। तो सोवेयर आसानी से बनाए रखने योग्य और एक्स्टेंसिबल होना चाहिए। एलएलडी सोवर के डिजाइन में एक महत्वपूर्ण भूमिका निभाता है और इसलिए सीधे सोवेयर की रखरखाव और विस्तारशीलता में योगदान देता है।

2. निम्न-स्तरीय डिज़ाइन साक्षात्कारों की तैयारी कैसे करें?

सोवेयर डेवलपर्स को न केवल साक्षात्कारों को क्रैक करने के लिए बल्कि मॉड्यूलर, एक्स्टेंसिबल, पुन: प्रयोज्य और रखरखाव योग्य सॉवेयर बनाने के लिए निम्न-स्तरीय डिज़ाइन सीखने की आवश्यकता है। यह विचार प्रक्रिया है जिसे डेवलपर्स को आवश्यकताओं के एक सेट से प्रभावी ढंग से सॉवर बनाने के लिए विकसित करने की आवश्यकता है।

यहां उन चीजों की सूची दी गई है जो एक सॉवेयर डेवलपर को एलएलडी साक्षात्कार के लिए सीखने की जरूरत है:

- ऑब्जेक्ट-ओरिएंटेड लैंग्वेज: मशीन कोडिंग राउंड (या) एलएलडी राउंड में सामान्य अपेक्षा यह है कि उम्मीदवार ऑब्जेक्ट ओरिएंटेड कोड लिखता है। जैसा कि नाम से पता चलता है, वस्तु-उन्मुख भाषाएँ वस्तुओं के बारे में हर चीज के इर्द-गिर्द घूमती हैं।
  उम्मीदवार को विभिन्न संस्थाओं की पहचान करनी चाहिए जिन्हें समस्या विवरण से बनाने की आवश्यकता है और उपयोग के मामले के आधार पर डेटा संरचनाओं के सही सेट को चुनने के साथ-साथ आवश्यक संस्थाओं के अनुसार कक्षाओं को मॉडल करना चाहिए।
  किसी भी वस्तु-उन्मुख भाषा को सीखना आवश्यक है क्योंकि उम्मीदवार को उस भाषा का उपयोग करके कोड लिखना होता है। कुछ लोकप्रिय ऑब्जेक्ट-ओरिएंटेड भाषाएं जावा, सी #, सी ++ और पायथन हैं।

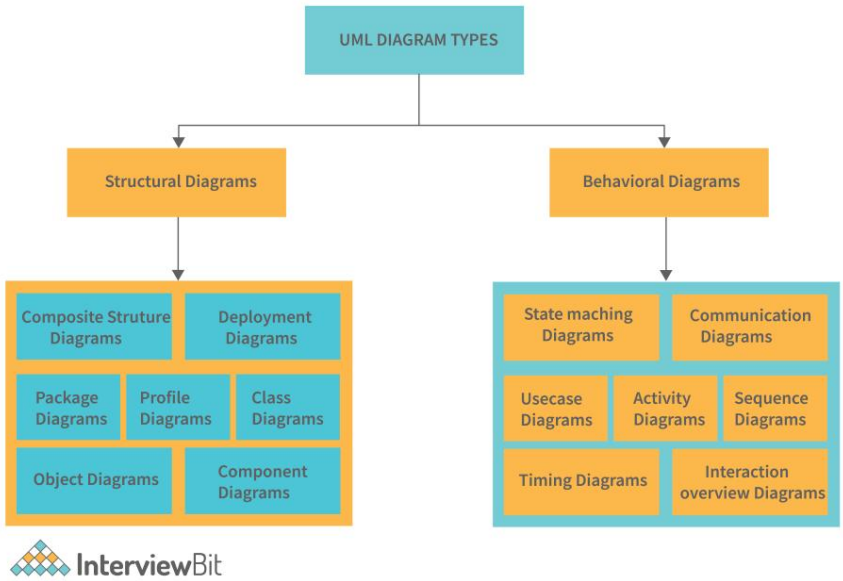
- ऑब्जेक्ट-ओरिएंटेड सिद्धांत: ऑब्जेक्ट-ओरिएंटेड भाषा को जानना ही एक्स्टेंसिबल और रखरखाव योग्य कोड लिखने के लिए पर्याप्त नहीं है। कई ऑब्जेक्ट-ओरिएंटेड सिद्धांतों को सोवेयर को एक्स्टेंसिबल और रखरखाव योग्य बनाने के लिए डिज़ाइन किया गया है।
  कुछ प्रसिद्ध सिद्धांत हैं:
  - YAGNI: आपको इसकी आवश्यकता नहीं है यह सोवेयर विकास में एक अभ्यास है जिसमें कहा गया है कि सुविधाओं को केवल आवश्यकता होने पर ही जोड़ा जाना चाहिए और इस प्रकार रिलीज की वांछित बढ़ी हुई आवृत्ति को सुविधाजनक बनाने के लिए अतिरिक्त और अक्षमता विकास को दूर करने में मदद मिलती है)

  - सूखा: अपने आप को दोहराएँ मत एक सिद्धांत है जो बताता है कि कोड को बार-बार न दोहराएं। यदि कोई कोड है जिसे डुप्लिकेट किया गया है तो जब भी कोई परिवर्तन करने की आवश्यकता होती है, तो परिवर्तन दोनों स्थानों पर करना होता है। इसलिए डेवलपर को उन सभी जगहों को याद रखना होगा जहां कोड डुप्लिकेट किया गया है जो कठिन और त्रुटि-प्रवण हैं। इस प्रकार यह सिद्धांत कहता है कि कोड को कभी भी डुप्लिकेट नहीं किया जाना चाहिए और इसकी विधि या वर्ग में रिफैक्टर करना पड़ता है और अन्य सभी स्थानों को कोड को डुप्लिकेट करने के बजाय इस विधि/वर्ग का उपयोग करने की आवश्यकता होती है।

  - ठोस: ये 5 सिद्धांतों का एक समूह है। एकल उत्तरदायित्व, खुला-बंद, लिस्कोव प्रतिस्थापन, इंटरफ़ेस अलगाव, और निर्भरता उलटा।

इनके अलावा, जाने-माने सिद्धांत हैं जैसे कि विरासत पर रचना का पक्ष लेना, जो भिन्न होता है उसे समाहित करना, शिथिल युग्मित वर्गों के लिए प्रयास करना आदि।

- यूएमएल आरेख: यूएमएल एक संक्षिप्त रूप है जो यूनिफाइड मॉडलिंग लैंग्वेज के लिए खड़ा है और सोवेयर सिस्टम की कलाकृतियों के मॉडलिंग, विज़ुअलाइज़ेशन और दस्तावेजीकरण के लिए एक मानक है। यह एक प्रोग्रामिंग भाषा नहीं है, बल्कि विभिन्न प्रकार के आरेखों को नेत्रहीन रूप से चित्रित करने का एक उपकरण है जो सोवर के निर्माण के लिए उपयोगी हो सकता है।
    - 14 विभिन्न प्रकार के यूएमएल आरेख हैं और इन 14 आरेखों को 2 उच्च-स्तरीय समूहों में वर्गीकृत किया गया है।
        - संरचनात्मक आरेख: वे प्रणाली के स्थिर दृष्टिकोण का प्रतिनिधित्व करते हैं। क्लास डायग्राम, कंपोजिट स्ट्रक्चर डायग्राम, ऑब्जेक्ट डायग्राम, कंपोनेंट डायग्राम, डिप्लॉयमेंट डायग्राम, प्रोफाइल डायग्राम और पैकेज डायग्राम स्ट्रक्चरल डायग्राम का हिस्सा हैं।
        - व्यवहार आरेख: वे प्रणाली के गतिशील दृष्टिकोण का प्रतिनिधित्व करते हैं। एक्टिविटी डायग्राम, स्टेट मशीन डायग्राम, यूज केस डायग्राम, इंटरेक्शन ओवरव्यू डायग्राम, टाइमिंग डायग्राम, सीक्वेंस डायग्राम और कम्युनिकेशन डायग्राम बिहेवियरल डायग्राम का हिस्सा हैं।



क्लास डायग्राम और यूज केस डायग्राम आमतौर पर एलएलडी राउंड में उपयोग किए जाते हैं।

- डिज़ाइन पैटर्न: डिज़ाइन पैटर्न सॉवर डिज़ाइन में सामान्य समस्याओं के विशिष्ट समाधान हैं। सॉवर इंजीनियरिंग में "डोंट रीइन्वेंट द व्हील" एक जाना-माना मुहावरा है। यदि कोई समस्या पहले ही हल हो चुकी है, तो उसी समस्या को हल करने के लिए उसी समस्या को हल न करें। इसके बजाय, समस्या के समाधान का उपयोग करें। इसी तरह, सॉवर डिज़ाइन में बहुत सी सामान्य समस्याएं हैं जिन्हें हल किया जाता है और उन्हें डिज़ाइन पैटर्न नामक टूलकिट के रूप में दिया जाता है। डिज़ाइन पैटर्न किसी विशेष भाषा के लिए विशिष्ट नहीं हैं। वे सामान्य समाधान हैं जो निर्दिष्ट करते हैं कि कैसे एक डिजाइन समस्या को हल किया जा सकता है और इसलिए इसे किसी भी भाषा में लागू किया जा सकता है। ये डिज़ाइन पैटर्न सिद्ध और परीक्षण किए गए हैं और इसलिए विकास प्रक्रिया को सुचारू और आसान बनाते हैं।

- एलएलडी प्रश्नों का अभ्यास करें: यथासंभव निम्न-स्तरीय डिज़ाइन साक्षात्कार प्रश्नों का अभ्यास करने का प्रयास करें। विभिन्न समस्याओं को हल करने से अलग-अलग दृष्टिकोण मिलते हैं और इस तरह डिजाइन कौशल में सुधार होता है। डेटा संरचनाओं और एल्गोरिथम समस्याओं के विपरीत डिज़ाइन समस्याओं का कोई सही समाधान नहीं है। तो विभिन्न समस्याओं को हल करने से अलग-अलग विचार मिलते हैं और इस प्रकार किसी भी डिजाइन समस्या को हल करने के लिए समग्र विचार प्रक्रिया विकसित होती है।

3. साक्षात्कार में निम्न-स्तरीय डिज़ाइन समस्याओं को कैसे हल करें?

एक साक्षात्कार में एलएलडी समस्या का समाधान आसानी से किया जा सकता है यदि हम समाधान को कई चरणों में विभाजित करते हैं और इन चरणों को एक-एक करके हल करने पर ध्यान केंद्रित करते हैं। एलएलडी समस्याओं को मोटे तौर पर 2 प्रकारों में वर्गीकृत किया जा सकता है: स्टैंडअलोन एप्लिकेशन और वेब एप्लिकेशन।

एलएलडी समस्या का समाधान मुख्य रूप से 3 चरणों में विभाजित किया जा सकता है:

1. स्पष्ट करें और आवश्यकताएँ इकट्ठा करें: निम्न-स्तरीय डिज़ाइन साक्षात्कार प्रश्न वास्तविक जीवन की स्थितियों के समान जानबूझकर ओपन-एंडेड / असंरचित होते हैं। समस्या के संबंध में प्रासंगिक प्रश्न पूछें (अच्छे प्रश्न वे हैं जो सिस्टम के व्यवहार, सुविधाओं और भविष्य में सिस्टम में कौन सी सुविधाओं को जोड़े जाने की उम्मीद है) के बारे में अधिक समझने में मदद करते हैं और सिस्टम की पूरी आवश्यकताओं को इकट्ठा करते हैं। पहले से कुछ भी ग्रहण न करें और साक्षात्कार के साथ हमेशा उन धारणाओं को स्पष्ट करें जिन्हें आप लेना चाहते हैं। आवश्यकताओं और मान्यताओं को लिखें और साक्षात्कारकर्ता के साथ उन पर चर्चा करें।

2. क्लास डायग्राम और यूज केस डायग्राम और स्कीमा डायग्राम (यदि आवश्यक हो): एक बार जरूरतें पूरी हो जाने के बाद, कोर क्लासेस और ऑब्जेक्ट्स को परिभाषित करें कि विभिन्न क्लासेस एक-दूसरे के साथ कैसे इंटरैक्ट करते हैं, और एक्टर्स जो सिस्टम के साथ इंटरैक्ट करते हैं, के लिए उपयोग के मामलों को परिभाषित करें। प्रत्येक अभिनेता, आदि। वर्ग आरेख बनाएं और सिस्टम के लिए केस आरेख (वैकल्पिक हो सकता है, साक्षात्कारकर्ता से पूछें) का उपयोग करें। कक्षाओं के बीच की बातचीत को देखकर उनके बीच संबंधों को परिभाषित करें। यदि क्लास डायग्राम और यूज केस डायग्राम सिस्टम की आवश्यकताओं और विभिन्न वर्गों के बीच संबंधों का प्रतिनिधित्व करने में मदद करता है, तो एक स्कीमा डायग्राम (जिसे ईआर डायग्राम भी कहा जाता है) डेटा मॉडल कैसे दिखता है और डेटाबेस में डेटा को कैसे स्टोर करता है, में मदद करता है। साक्षात्कारकर्ता डेटा को वास्तविक डेटाबेस में संग्रहीत करने के लिए कोड लिखने के लिए नहीं कह सकते हैं, लेकिन वे इस बात में रुचि ले सकते हैं कि मॉडल कैसा दिखता है और उनके बीच संबंध कैसा है। कक्षा आरेख से स्कीमा आरेख बनाना आसान है। वर्ग आरेख में प्रत्येक वर्ग स्कीमा आरेख में एक तालिका बन जाता है और वर्गों के बीच संबंध तालिकाओं के बीच बहुलता बन जाता है।

3. कोड: अंत में एक बार क्लास डायग्राम, यूज़ केस डायग्राम और स्कीमा डायग्राम (यदि आवश्यक हो) का उपयोग करके विचारों को संरचित किया जाता है, तो उम्मीदवार कोड लिखना शुरू कर सकता है।
सिस्टम को पुन: प्रयोज्य, एक्स्टेंसिबल और रखरखाव योग्य बनाने के लिए जहां भी संभव हो, डिज़ाइन पैटर्न, ऑब्जेक्ट-ओरिएंटेड सिद्धांतों और ठोस सिद्धांतों को लागू करें। सुनिश्चित करें कि कोड अच्छी तरह से संरचित है और कक्षाओं और पद्धति को साफ करने के लिए स्वच्छ कोडिंग प्रथाओं का पालन करें। डिज़ाइन पैटर्न को कोड में फ़िट करने का प्रयास न करें, लेकिन जाँच करें कि क्या किसी उपलब्ध डिज़ाइन पैटर्न का उपयोग करके दी गई समस्या को हल किया जा सकता है। उपरोक्त सभी बातों का ध्यान रखते हुए कोड की पठनीयता का ध्यान रखें। हां, सॉवर इंजीनियरिंग कठिन है।

4. सांप और सीढ़ी को कैसे डिजाइन करें?

यहां बताया गया है कि उपरोक्त दृष्टिकोण का उपयोग करके "सांप और सीढ़ी की समस्या" को कैसे हल किया जा सकता है।

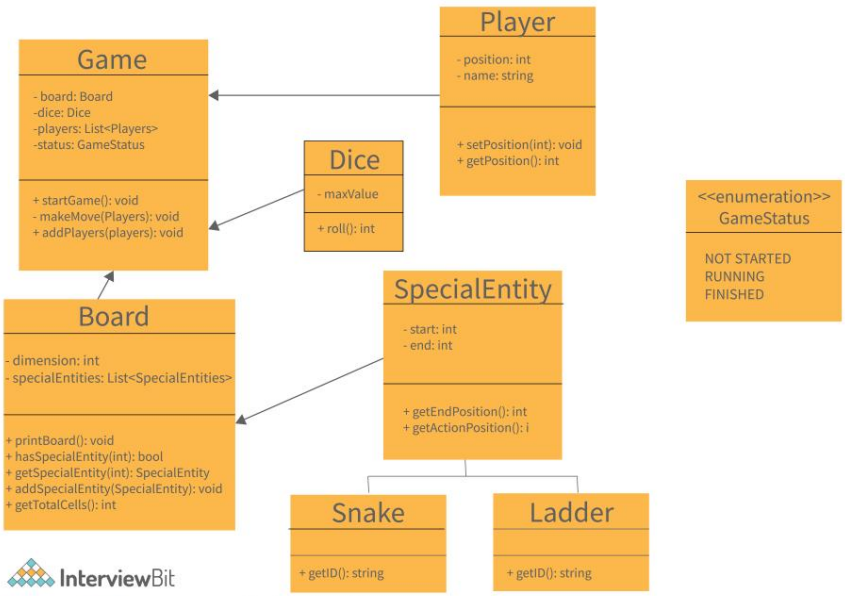
1. Gathering Requirements: Create a multiplayer Snake and Ladder game in such a way that the game should take input N from the user at the start of the game to create a board of size N x N. The board should have some snakes and ladders placed randomly in such a way the snake will have its head position at a higher number than the tail and ladder will have start position at smaller number than end position. Also, No ladder and snake create a cycle and no snake tail position has a ladder start position & vice versa.

The players take their turn one after another and during their turn, they roll the dice to get a random number and the player has to move forward that many positions. If the player ends up at a cell with the head of the snake, the player has to be punished and should go down to the cell that contains the tail of the same snake & If the player ends up at a cell with the start position of a ladder, the player has to be rewarded and should climb the ladder to reach the cell which has a top position of the ladder. Initially, each player is outside of the board (at position 0). If at any point of time, the player has to move outside of the board (say player at position 99 on a 100 cell board and dice rolls give 4) the player stays at the same position.

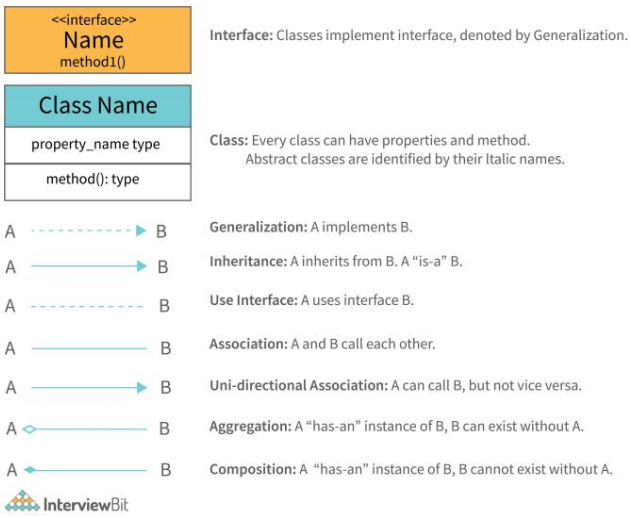
Possible future extension:

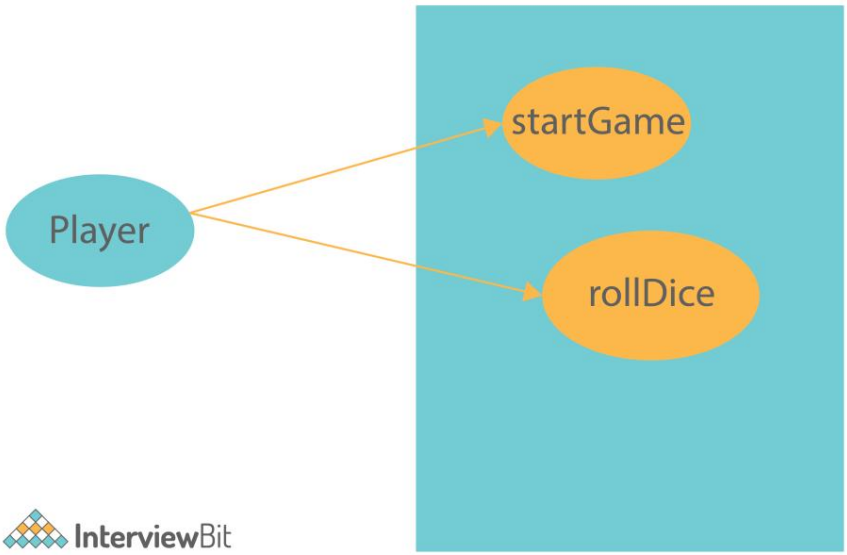
- The game can be played by more than one dice. (i.e. if there are two dices then the numbers from 2 to 12 will be generated).
- On getting a 6, you get another turn and on getting 3 consecutive 6s, all three of those get cancelled.

2. Class Diagram & Use Case Diagram:



UML CONVENTIONS





3. कोड कार्यान्वयन:

ड्राइवर.जावा

```
पैकेज स्नेकलैडर;


पब्लिक क्लास ड्राइवर {

        सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग [] तर्क) {

                स्पेशल एंटिटी स्नेक1 = नया स्नेक(12, 28);
                स्पेशल एंटिटी सांप 2 = नया सांप (34, 78);
                स्पेशल एंटिटी स्नेक3 = नया सांप (6, 69);
                स्पेशल एंटिटी स्नेक 4 = नया सांप (65, 84);

                SpecialEntity सीढ़ी1 = नई सीढ़ी (24, 56);
                SpecialEntity सीढ़ी2 = नई सीढ़ी(43, 83);
                SpecialEntity सीढ़ी3 = नई सीढ़ी(3, 31);
                SpecialEntity सीढ़ी4 = नई सीढ़ी(72, 91);

                बोर्ड बोर्ड = नया बोर्ड(10);
                Board.addSpecialEntity(snake1);
                Board.addSpecialEntity(snake2);
                Board.addSpecialEntity(snake3);
                Board.addSpecialEntity(snake4)

                Board.addSpecialEntity (सीढ़ी 1);
                Board.addSpecialEntity (सीढ़ी 2);
                Board.addSpecialEntity(सीढ़ी 3);
                Board.addSpecialEntity (सीढ़ी 4);

                पासा पासा = नया पासा (6);

                गेम गेम = नया गेम (बोर्ड, पासा);

                प्लेयर प्लेयर 1 = नया प्लेयर ("पी 1");
                प्लेयर प्लेयर 2 = नया प्लेयर ("पी 2");
                प्लेयर प्लेयर 3 = नया प्लेयर ("पी 3");

                खिलाड़ी = सूची <खिलाड़ी> () {खिलाड़ी 1, खिलाड़ी 2, खिलाड़ी 3}; game.addPlayers
                (खिलाड़ी);

                खेल। लॉन्च ();
        }

}
```

गेम.जावा

```
पैकेज स्नेकलैडर;

सांपलाडर आयात करें। गेम स्थिति; सांपलाडर आयात करें।
खिलाड़ी; सांपलाडर आयात करें। पासा; सांपलाडर आयात
करें।बोर्ड;


आयात java.util.Queue; आयात
java.util.Scanner; आयात
java.util.LinkedList;

पब्लिक क्लास गेम {

        बोर्ड बोर्ड;
        पासा पासा;
        कतार <खिलाड़ी> खिलाड़ी;
        गेमस्टैटस स्टॉस;

        सार्वजनिक खेल (बोर्ड बोर्ड, पासा पासा) {

                यह बोर्ड = बोर्ड; यह पासा = पासा;
                this.players = new
                LinkedList<Player>(); this.status = GameStatus.NOT_STARTED;

        }

        सार्वजनिक शून्य स्टार्टगेम () {

                this.status = GameStatus.RUNNING; बोर्ड.प्रिंटबोर्ड
                ();

                // तब तक चलाएं जब तक हमारे पास बोर्ड पर केवल 1 खिलाड़ी बचा हो (खिलाड़ियों का आकार ()>
                1) {

                        प्लेयर प्लेयर = प्लेयर्स पोल ();

                        मेकमोव (कर्लप्लेयर);

                        अगर (खिलाड़ी। getPosition () == बोर्ड। getTotalCells ())
                                System.out.println(player.getName() + "खेल पूरा कर लिया है!"); अन्य खिलाड़ी। जोड़ें (खिलाड़ी);


                }

                this.status = GameStatus.FINISHED;

        }

        निजी शून्य मेकमूव (खिलाड़ी खिलाड़ी) {

                System.out.println ();
                System.out.println(currPlayer.getUserName()+"'s बारी।"); System.out.println ("
                पासा रोल करने के लिए कुछ भी दबाएं।");

                स्कैनर एससी = नया स्कैनर (System.in); चार सी = एससी.अगला
                ()। चारएट (0);

                मैं _ द्वितीय मैं द्वितीय ()
```

बोर्ड जावा

```
पैकेज स्नेकलैडर;

आयात लोम्बोक। गेट्टर;

पब्लिक क्लास बोर्ड{

        @ गेट्टर इंट
        आयाम;

        हैश मैप <इंटीजर, स्पेशल एंटिटी> स्पेशल एंटिटीज;

        सार्वजनिक बोर्ड (इंट आयाम) {

                यह आयाम = आयाम;
        }

        सार्वजनिक शून्य प्रिंटबोर्ड () {

                int TotalCells = आयाम * आयाम; for(int i=totalCells; i > 0;
                i--) {

                        सिस्टम.आउट.प्रिंट (" |"                    + मैं +       " ")

                        अगर (है विशेष इकाई (i))
                                System.out.print(specialEntities.get(i).getID());

                        सिस्टम.आउट.प्रिंट (" |"); अगर (कुल
                        सेल% 10 == 0)
                                System.out.println ();
                }
        }

        सार्वजनिक int getTotalCells () {

                इसे वापस करें। आयाम * यह। आयाम;
        }

        सार्वजनिक शून्य addSpecialEntity(SpecialEntity इकाई) {

                int actionPosition = entity.getActionPosition ();

                SpecialEntities.put (कार्रवाई स्थिति, इकाई);
        }

        पब्लिक बूलियन में स्पेशल एंटिटी (इंट पोजीशन) {

                विशेष वापसी करें। शामिल हैंकी (स्थिति);
        }

        public SpecialEntity getSpecialEntity(int position) {

                if(hasSpecialEntity(position)) return
                        specialEntities.get(position);

                वापसी शून्य;
        }
}
```

पासा.जावा

```
पैकेज स्नेकलैडर;

आयात लोम्बोक। गेट्टर;

पब्लिक क्लास डाइस{

        इंट मैक्सवैल्यू;

        सार्वजनिक पासा (इंट मैक्सवैल) {

                this.maxValue = maxVal;
        }

        सार्वजनिक इंट रोल () {

                वापसी (int) Math.floor(Math.random()*maxValue + 1);
        }
}
```

GameStatus.java

```
पैकेज स्नेकलैडर;

सार्वजनिक एनम गेमस्टैटस {
        शुरू नहीं,
        दौड़ना,
        खत्म
}
```

SpecialEntity.java

```
पैकेज स्नेकलैडर;

आयात लोम्बोक। गेट्टर; सार्वजनिक अमूर्त
वर्ग बोर्ड इकाई {

        निजी int प्रारंभ; निजी अंतर अंत;


        सार्वजनिक विशेष इकाई (इंट स्टार्ट, इंट एंड) {
                यह। प्रारंभ = प्रारंभ; यह अंत = अंत;

        }

        सार्वजनिक सार स्ट्रिंग getID ();

        सार्वजनिक int getActionPosition () {

                इसे वापस करें। प्रारंभ करें;
        }

        सार्वजनिक int getEndPosition () {

                इसे वापस करो।अंत;
        }
}
```

नाग.जावा

```
पैकेज स्नेकलैडर;

पब्लिक क्लास स्नेक स्पेशलएंटिटी का विस्तार करता है{

        पब्लिक स्नेक (इंट स्टार्ट, इंट एंड) { सुपर (स्टार्ट, एंड);

        }

        @Override
        सार्वजनिक स्ट्रिंग getID () { वापसी "S_"
                + this.getEnd ();
        }
}
```

सीढ़ी.जावा

```
पैकेज स्नेकलैडर;

पब्लिक क्लास लैडर स्पेशलएन्टिटी का विस्तार करता है {

        सार्वजनिक सीढ़ी (इंट स्टार्ट, इंट एंड) { सुपर (स्टार्ट, एंड);

        }

        @ ओवरराइड
        सार्वजनिक स्ट्रिंग getID () { वापसी "L_"
                + this.getEnd ();
        }
}
```

प्लेयर.जावा

```
पैकेज स्नेकलैडर;

आयात लोम्बोक। गेट्टर; आयात लोम्बोक।
सेटर;

पब्लिक क्लास प्लेयर{

        @ गेट्टर
        @सेटर इंट
        पोजीशन;

        @ गेट्टर
        स्ट्रिंग नाम;

        सार्वजनिक खिलाड़ी (स्ट्रिंग नाम) {

                यह नाम = नाम; यह स्थिति = 0;

        }
}
```

गेमपहले से ही शुरू हो गयाअपवाद.जावा

```
पैकेज स्नेकलैडर;

सार्वजनिक वर्ग GameAlreadyStartedException अपवाद बढ़ाता है { सार्वजनिक
        GameAlreadyStartedException (स्ट्रिंग संदेश) {
                सुपर (संदेश);
        }
}
```

कोड को अलग-अलग वर्गों में संरचित करना महत्वपूर्ण है जिसमें प्रत्येक वर्ग की एक ही जिम्मेदारी होती है (SOLID सिद्धांतों में S को याद रखें) और कक्षाओं के बीच संबंध भी होना चाहिए ताकि वे आसानी से एक्स्टेंसिबल हो सकें।

अभ्यास करने के लिए अक्सर पूछे जाने वाले निम्न-स्तरीय डिज़ाइन उदाहरण हैं:

- डिजाइन पार्किंग लॉट
- डिजाइन स्प्लिटवाइज
- डिजाइन टिक टोक टो गेम
- डिजाइन कार रेंटल सिस्टम
- डिजाइन बुकमायशो
- डिजाइन पब उप प्रणाली
- डिजाइन कॉफी वेंडिंग मशीन

लो लेवल डिज़ाइन (LLD) को क्रैक करने के टिप्स साक्षात्कार

5. इंटरव्यू की तैयारी के टिप्स

- जल्दी मत करो: एक बार आवश्यकताओं को सूचीबद्ध करने के बाद, विचार करने के लिए बहुत अधिक आवश्यकताएं हो सकती हैं और 40 या 45 मिनट डिजाइन और कोड के लिए पर्याप्त नहीं हो सकते हैं। सभी आवश्यकताओं के लिए डिज़ाइन, और कोड को पूरा करने में जल्दबाजी न करें। साक्षात्कारकर्ता के साथ चर्चा करें और केवल उन मुख्य आवश्यकताओं का उल्लेख करें जिन पर आप डिजाइन के लिए विचार करना चाहते हैं। दौड़कर दूध पीने से अच्छा है खड़े होकर पानी पीना।

- अभ्यास करें: एक निम्न-स्तरीय डिज़ाइन समस्या निकालें और उन सभी को स्वयं हल करने का प्रयास करें। आवश्यकताओं को इकट्ठा करें (यहां अपने न्यायाधीश बनें), वर्ग और स्कीमा आरेख बनाएं और कोड लिखें जो एक्स्टेंसिबल, पुन: प्रयोज्य और रखरखाव योग्य है। एक बार जब आप समस्या को हल कर लेते हैं, तो इंटरनेट पर एक ही समस्या के लिए अलग-अलग समाधान देखें और कमजोरी में सुधार करें। इंटरव्यू से पहले जितनी हो सके उतनी समस्याओं का अभ्यास करें। यदि आप भाग्यशाली हैं, तो आपको साक्षात्कार में पहले से हल की गई समस्या भी मिल सकती है। यदि नहीं, तो आपके पास पहले से ही किसी नई समस्या को हल करने के लिए एक विचार प्रक्रिया विकसित की गई है। जीत-जीत की स्थिति प्रत्येक चरण के लिए योजना: दुर्भाग्य से, दी गई समस्या को हल करने के लिए पर्याप्त नहीं है, हमें इसे समय सीमा के भीतर पूरा करने की भी आवश्यकता है। चूंकि, हमारे पास निम्न-स्तरीय
- डिज़ाइन साक्षात्कार प्रश्न को हल करने के लिए अलग-अलग चरण हैं जैसे कि आवश्यकताओं को इकट्ठा करना, वर्ग आरेख, स्कीमा आरेख, कोड, परीक्षण, आदि। योजना बनाएं कि आप प्रत्येक चरण पर कितना प्रतिशत साक्षात्कार समय व्यतीत करना चाहते हैं। जितना अधिक आप अभ्यास करते हैं, आप प्रत्येक चरण में आपके द्वारा लिए जा रहे समय का विश्लेषण भी कर सकते हैं और जांच सकते हैं कि आपको कहां सुधार करने की आवश्यकता है।

- टेस्टकेस लिखें और अपवादों को संभालें: प्रत्येक साक्षात्कारकर्ता आपसे कोड के लिए टेस्ट केस लिखने की अपेक्षा नहीं करता है। लेकिन अगर आप एक कदम आगे जाकर परीक्षण के मामलों को लिख सकते हैं, तो यह आपको अन्य उम्मीदवारों पर एक बड़ा लाभ देता है। अपवादों और अन्य कोने के मामलों को गहनता से संभालें।

- टूल के साथ सहज हो जाएं: जैसा कि हमें क्लास डायग्राम, स्कीमा डायग्राम, केस डायग्राम आदि का उपयोग करने, विभिन्न घटकों को डिजाइन करने और उनके बीच उनकी बातचीत की आवश्यकता हो सकती है, यह सामान्य है कि कंपनियां व्हाइटबोर्ड लिंक और कोड एडिटर के साथ अपने डिजाइन साक्षात्कार को शेड्यूल करती हैं। व्हाइटबोर्ड ऐप्स/कोड संपादकों के आदी हुए बिना उनका उपयोग करना मुश्किल हो सकता है। इसलिए साक्षात्कार से पहले इन उपकरणों के साथ सहज हो जाएं ताकि आपको यह पता लगाने में समय बर्बाद न करना पड़े कि इन उपकरणों के साथ कैसे काम करना है।

निष्कर्ष

इस लेख में, हमने चर्चा की कि निम्न-स्तरीय डिज़ाइन क्या है और यह उच्च-स्तरीय डिज़ाइन साक्षात्कारों से कैसे भिन्न है। निम्न-स्तरीय डिज़ाइन किसी एप्लिकेशन के वर्ग-स्तरीय डिज़ाइन और स्वच्छ, पठनीय, रखरखाव योग्य और एक्स्टेंसिबल कोड पर केंद्रित है। निश्चित आवश्यकताओं के लिए कोई सोवेयर नहीं बनाया गया है और जैसे-जैसे सोवर विकसित होगा, नई आवश्यकताएं होंगी और इसलिए सोवर को बदलने की आवश्यकता है। सॉफ़्टवेयर विकास में परिवर्तन ही एकमात्र स्थिरांक है। इसलिए सॉवेयर का डिजाइन अच्छा होना चाहिए। निम्न-स्तरीय डिज़ाइन साक्षात्कारों को क्रैक करने की कुंजी वास्तविक दुनिया के अनुप्रयोगों के विकास में ऑब्जेक्ट ओरिएंटेड सिद्धांतों और डिज़ाइन पैटर्न के महत्व और उपयोग को समझना है और कैसे ये अवधारणाएं एप्लिकेशन को पठनीय, रखरखाव योग्य और एक्स्टेंसिबल बनाती हैं।

निम्न-स्तरीय डिज़ाइन समस्या को स्पष्ट रूप से समझें, समस्या की आवश्यकताओं को संक्षेप में लिखें, आवश्यकताओं को ठीक करें और मूल आवश्यकताओं को चुनें जिन्हें दिए गए समय में लागू किया जा सकता है, विभिन्न संस्थाओं को उनके व्यवहार और मुख्य आवश्यकताओं से अभिनेताओं के साथ नोट करें , उपयोग केस आरेख और वर्ग आरेख बनाएं, स्कीमा आरेख बनाएं यदि समस्या विवरण के लिए एक प्रकार के वेब एप्लिकेशन का निर्माण करना आवश्यक है (साक्षात्कार में वेब एप्लिकेशन बनाने की आवश्यकता नहीं है लेकिन नियंत्रक बनाकर कोड को उस तरह से संरचित किया जा सकता है , सेवाओं, और रिपॉजिटरी) और फिर अंत में एप्लिकेशन को एक्स्टेंसिबल और पठनीय बनाने के लिए ऑब्जेक्ट ओरिएंटेड प्रिंसिपल्स और डिज़ाइन पैटर्न का उपयोग करके मुख्य विशेषताओं को लागू करना।

यथासंभव निम्न-स्तरीय डिज़ाइन साक्षात्कार प्रश्नों का अभ्यास करें क्योंकि प्रत्येक समस्या एक अलग परिप्रेक्ष्य देती है और समस्याओं को हल करने के लिए ऑब्जेक्ट ओरिएंटेड सिद्धांतों और डिज़ाइन पैटर्न को लागू करना सीखें।

साक्षात्कार संसाधन

- सिस्टम डिजाइन साक्षात्कार प्रश्न
- सॉफ्टवेयर इंजीनियरिंग साक्षात्कार प्रश्न
- बेस्ट सिस्टम डिजाइन कोर्स
- डिजाइन पैटर्न साक्षात्कार प्रश्न
- जावा साक्षात्कार के प्रश्न
- OOPs साक्षात्कार प्रश्न
- साक्षात्कार तैयारी संसाधन

अधिक साक्षात्कार के लिए लिंक प्रश्न

ग साक्षात्कार सवाल

पी एच पी सम्बंदित इन्टर्व्यू के सवाल

सी तीव्र साक्षात्कार प्रश्न

वेब एपीआई साक्षात्कार प्रश्न

हाइबरनेट साक्षात्कार प्रश्न

नोड जेएस साक्षात्कार प्रश्न

सीपीपी साक्षात्कार प्रश्न

उफ़ साक्षात्कार प्रश्न

देवोप्स साक्षात्कार प्रश्न

मशीन लर्निंग इंटरव्यू प्रश्न

डॉकर साक्षात्कार प्रश्न मैसकल साक्षात्कार प्रश्न

सीएसएस साक्षात्कार प्रश्न

लारवेल साक्षात्कार प्रश्न एएसपी नेट साक्षात्कार प्रश्न

Django साक्षात्कार प्रश्न डॉट नेट साक्षात्कार प्रश्न कुबेरनेट्स साक्षात्कार प्रश्न

ऑपरेटिंग सिस्टम साक्षात्कार प्रश्न

प्रतिक्रिया मूल निवासी साक्षात्कार प्रश्न

एडब्ल्यूएस साक्षात्कार प्रश्न

गिट साक्षात्कार प्रश्न

जावा 8 साक्षात्कार प्रश्न मोंगोडब साक्षात्कार प्रश्न

डीबीएमएस साक्षात्कार सवाल

स्प्रिंग बूट साक्षात्कार प्रश्न

पावर बीआई साक्षात्कार प्रश्न

Pl Sql साक्षात्कार प्रश्न

झांकी साक्षात्कार प्रश्न

लिनक्स साक्षात्कार प्रश्न

उत्तरदायी साक्षात्कार प्रश्न जावा साक्षात्कार प्रश्न

जेनकींस साक्षात्कार प्रश्न