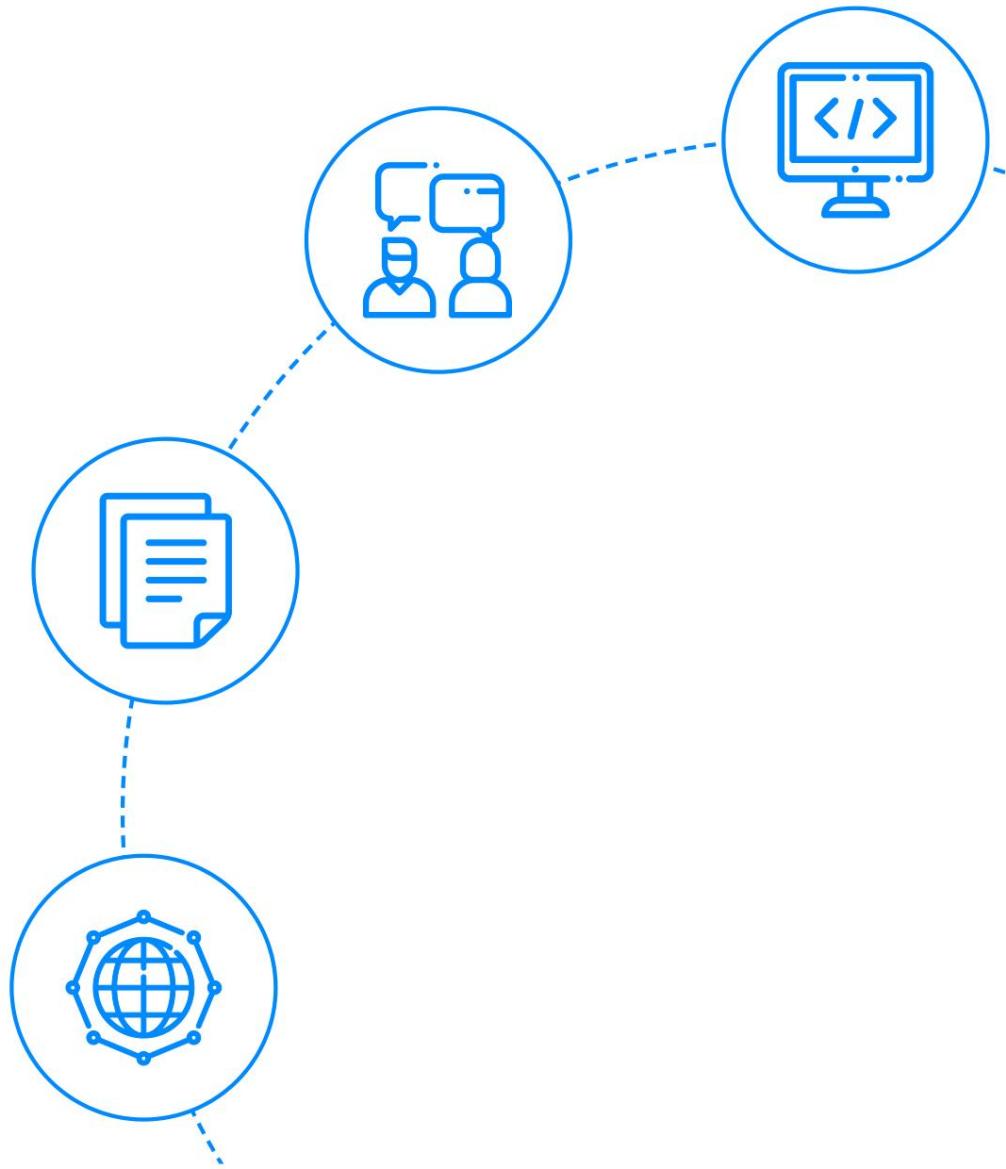




जावा साक्षात्कार के प्रश्न



का लाइव संस्करण देखने के लिए
पेज, यहां [क्लिक करें।](#)

अंतर्वस्तु

जावा बेसिक साक्षात्कार प्रश्न

1. जावा एक मंच स्वतंत्र भाषा क्यों है?
2. जावा शुद्ध वस्तु उन्मुख भाषा क्यों नहीं है?
3. पॉइंटर्स का उपयोग C/C++ में किया जाता है। जावा पॉइंटर्स का उपयोग क्यों नहीं करता है?
4. एक आवृत्ति चर और एक स्थानीय चर से आप क्या समझते हैं?
5. डेटा एनकैप्सुलेशन से आप क्या समझते हैं?
6. हमें जेआईटी कंपाइलर के बारे में कुछ बताएं।
7. क्या आप बराबर () विधि और समानता ऑपरेटर (==) के बीच अंतर बता सकते हैं जावा?
8. जावा में अनंत लूप कैसे घोषित किया जाता है?
9. कंस्ट्रक्टर ओवरलोडिंग की अवधारणा को संक्षेप में समझाएं
10. प्रासंगिक उदाहरणों का हवाला देते हुए विधि ओवरलोडिंग और ओवरराइडिंग पर टिप्पणी करें।
11. जावा प्रोग्राम में सिंगल ट्राई ब्लॉक और मल्टीपल कैच ब्लॉक एक साथ मौजूद हो सकते हैं। समझाना।
12. वेरिएबल, मेथड और क्लास में फाइनल कीवर्ड के उपयोग की व्याख्या करें।
13. क्या अंतिम, अंतिम और अंतिम कीवर्ड का कार्य समान है?
14. आप सुपर कीवर्ड का उपयोग कब कर सकते हैं?
15. क्या स्थैतिक विधियों को अतिभारित किया जा सकता है?
16. क्या स्थैतिक विधियों को ओवरराइड किया जा सकता है?
17. कचरा संग्रहण का मुख्य उद्देश्य क्या है?
18. मेमोरी का कौन सा भाग - स्टैक या हीप - कचरा संग्रहण प्रक्रिया में साफ किया जाता है?

जावा इंटरमीडिएट साक्षात्कार प्रश्न

जावा इंटरमीडिएट साक्षात्कार प्रश्न (.....जारी)

19. सुरक्षा पहलू के अलावा, जावा में स्ट्रिंग्स को अपरिवर्तनीय बनाने के पीछे क्या कारण हैं?
20. आप स्ट्रिंग, स्ट्रिंगबफर और स्ट्रिंगबिल्डर के बीच अंतर कैसे करेंगे?
21. प्रासंगिक गुणों का उपयोग इंटरफेस के बीच अंतर को उजागर करता है और अमूर्त वर्ग।
22. जावा में, स्थिर और साथ ही निजी पद्धति को ओवरराइड करना संभव है। पर टिप्पणी करें बयान।
23. क्या हैशसेट को ट्रीसेट से अलग बनाता है?
24. गोपनीय जानकारी संग्रहीत करने के लिए वर्ण सरणी को स्ट्रिंग पर क्यों पसंद किया जाता है?
25. जावा में JVM, JRE और JDK में क्या अंतर हैं?
26. जावा में हैश मैप और हैशटेबल के बीच क्या अंतर हैं?
27. जावा में प्रतिबिंब का क्या महत्व है?
28. धारे के उपयोग के विभिन्न तरीके क्या हैं?
29. जावा में क्लास के कंस्ट्रक्टर और मेथड में क्या अंतर हैं?
30. जावा "पास बाय वैल्यू" या "पास बाय रेफरेंस" घटना के रूप में काम करता है?
31. स्ट्रिंग या स्ट्रिंग बफर में से कौन सा पसंद किया जाना चाहिए जब बहुत सारे डेटा में अद्यतन करने की आवश्यकता है?
32. जावा में किसी वर्ग की विशेषताओं के क्रमांकन की अनुमति कैसे न दें?
33. यदि जावा में मुख्य विधि हस्ताक्षर में स्थिर संशोधक शामिल नहीं है तो क्या होगा?
34. क्या होता है यदि जावा में एक वर्ग के अंदर कई मुख्य विधियाँ हैं?
35. ऑब्जेक्ट क्लोनिंग से आप क्या समझते हैं और आप इसे जावा में कैसे प्राप्त करते हैं?
36. कोड में अपवाद कैसे फैलता है?
37. क्या कैच ब्लॉक के लिए ट्राई ब्लॉक का पालन करना अनिवार्य है?
38. क्या ट्राई ब्लॉक और कैच ब्लॉक के अंत में रिटर्न स्टेटमेंट लिखे जाने पर अंत में ब्लॉक निष्पादित हो जाएगा जैसा कि नीचे दिखाया गया है?

जावा इंटरव्यूडिएट साक्षात्कार प्रश्न (.....जारी)

39. क्या आप किसी क्लास के कंस्ट्रक्टर को दूसरे कंस्ट्रक्टर के अंदर बुला सकते हैं?
40. सन्निहित स्मृति स्थानों का उपयोग आमतौर पर किसी सरणी में वास्तविक मानों को संग्रहीत करने के लिए किया जाता है, लेकिन ArrayList में नहीं। समझाना।

जावा उन्नत साक्षात्कार प्रश्न

41. हालांकि वंशानुक्रम एक लोकप्रिय ओओपी अवधारणा है, यह संरचना से कम फायदेमंद है। समझाना।

42. नए () का उपयोग करके एक स्ट्रिंग का निर्माण एक शाब्दिक से अलग कैसे है?

43. क्या कूड़ा-करकट होने के बावजूद प्रोग्राम में मेमोरी लिमिट को पार करना संभव है?
एकत्र करनेवाला?

44. सिंक्रनाइज़ेशन क्यों आवश्यक है? प्रासंगिक उदाहरण की सहायता से स्पष्ट कीजिए।

45. नीचे दिए गए कूट में... का क्या महत्व है?

46. क्या आप जावा थ्रेड जीवनचक्र की व्याख्या कर सकते हैं?

47. एक अनियंत्रित सरणी बनाम के उपयोग के बीच ट्रेडऑफ क्या हो सकता है
एक आदेशित सरणी का उपयोग?

48. क्या जावा में एक ही वर्ग या पैकेज को दो बार आयात करना संभव है और रनटाइम के दौरान इसका क्या होता है?

49. यदि किसी पैकेज में उप पैकेज हैं, तो क्या यह केवल मुख्य पैकेज का आयात करने के लिए पर्याप्त होगा? उदाहरण के लिए क्या com.myMainPackage.* का आयात भी com.myMainPackage.mySubPackage.* आयात करता है?

50. यदि कोड System.exit(0) अंत में लिखा है तो क्या अंत में ब्लॉक निष्पादित किया जाएगा?
कोशिश ब्लॉक की?

51. जावा में मार्कर इंटरफेस से आप क्या समझते हैं?

52. जावा में "डबल ब्रेस इनिशियलाइज़ेशन" शब्द की व्याख्या करें?

53. ऐसा क्यों कहा जाता है कि स्ट्रिंग वर्ग की लंबाई () विधि सटीक नहीं लौटती है
परिणाम?

54. नीचे दिए गए कोड का आउटपुट क्या है और क्यों?

55. कचरा संग्रहण (जीसी) के लिए वस्तु को योग्य बनाने के संभावित तरीके क्या हैं
जावा में?

जावा साक्षात्कार कार्यक्रम

(..... जारी)

56. रिकर्सन का उपयोग करके जांचें कि क्या दिया गया स्ट्रिंग पैलिंड्रोम है।
57. यह जांचने के लिए जावा प्रोग्राम लिखें कि क्या दो तार विपर्यय हैं।
58. किसी दी गई संख्या का भाज्य ज्ञात करने के लिए जावा प्रोग्राम लिखें।
59. 1 से n तक की गैर-डुप्लिकेटिंग संख्याओं की एक सरणी को देखते हुए, जहां एक संख्या गायब है, उस लापता संख्या को खोजने के लिए एक कुशल जावा प्रोग्राम लिखें।
60. यह जांचने के लिए जावा प्रोग्राम लिखें कि कोई संख्या जादुई संख्या है या नहीं। ए संख्या को एक जादुई संख्या कहा जाता है यदि प्रत्येक चरण में अंकों का योग करता है और उस योग के अंकों का योग करता है, तो अंतिम परिणाम (जब केवल एक अंक |e होता है) 1 होता है।

निष्कर्ष



61. निष्कर्ष

आएँ शुरू करें

क्या आपके पास जावा साक्षात्कार में सफल होने के लिए क्या आवश्यक है? हम यहां जावा में आपके ज्ञान और अवधारणाओं को समेकित करने में आपकी सहायता करने के लिए हैं। निम्नलिखित लेख में फ्रेशर्स के साथ-साथ अनुभवी उम्मीदवारों के लिए सभी लोकप्रिय जावा साक्षात्कार प्रश्नों को गहराई से शामिल किया जाएगा।

साक्षात्कार में अच्छा प्रदर्शन करने की संभावनाओं को बढ़ाने के लिए सभी प्रश्नों को पढ़ें। प्रश्न जावा के मूल और मूल सिद्धांतों के इर्द-गिर्द घूमेंगे।

तो, आइए जावा पर उपयोगी साक्षात्कार प्रश्नों के ढेरों पर गहराई से विचार करें।

जावा बेसिक साक्षात्कार प्रश्न

1. जावा एक मंच स्वतंत्र भाषा क्यों है?

जावा भाषा को इस तरह से विकसित किया गया था कि यह किसी भी हार्डवेयर या सॉफ्टवर पर निर्भर नहीं करता है क्योंकि संकलक कोड को संकलित करता है और फिर इसे प्लेटफॉर्म-स्वतंत्र बाइट कोड में परिवर्तित करता है जिसे कई सिस्टम पर चलाया जा सकता है।

- उस बाइट कोड को चलाने के लिए एकमात्र शर्त यह है कि मशीन में रनटाइम वातावरण (JRE) स्थापित हो।

2. जावा शुद्ध वस्तु उन्मुख भाषा क्यों नहीं है?

जावा आदिम डेटा प्रकारों का समर्थन करता है - बाइट, बूलियन, चार, शॉर्ट, इंट, फ्लोट, लॉन्ग और डबल और इसलिए यह शुद्ध ऑब्जेक्ट-ओरिएंटेड भाषा नहीं है।

3. पॉइंटर्स का उपयोग C/C++ में किया जाता है। जावा का उपयोग क्यों नहीं करता संकेत?

शुरुआती प्रोग्रामर द्वारा उपयोग करने के लिए पॉइंटर्स काफी जटिल और असुरक्षित हैं। जावा कोड सादगी पर ध्यान केंद्रित करता है, और पॉइंटर्स का उपयोग इसे चुनौतीपूर्ण बना सकता है। सूचक उपयोग भी संभावित त्रुटियों का कारण बन सकता है। इसके अलावा, यदि पॉइंटर्स का उपयोग किया जाता है तो सुरक्षा से भी समझौता किया जाता है क्योंकि उपयोगकर्ता पॉइंटर्स की मदद से सीधे मेमोरी तक पहुंच सकते हैं।

इस प्रकार, जावा में पॉइंटर्स को शामिल न करके अमूर्तता का एक निश्चित स्तर प्रस्तुत किया जाता है। इसके अलावा, पॉइंटर्स का उपयोग कचरा संग्रहण की प्रक्रिया को काफी धीमा और गलत बना सकता है। जावा संदर्भों का उपयोग करता है क्योंकि पॉइंटर्स के विपरीत इन्हें हेरफेर नहीं किया जा सकता है।

4. इंस्टेंस वेरिएबल और लोकल से आप क्या समझते हैं? चर?

इंस्टेंस वेरिएबल वे वेरिएबल हैं जो कक्षा में सभी विधियों द्वारा सुलभ हैं। उन्हें विधियों के बाहर और कक्षा के अंदर घोषित किया जाता है। ये चर किसी वस्तु के गुणों का वर्णन करते हैं और किसी भी कीमत पर उससे बंधे रहते हैं।

कक्षा की सभी वस्तुओं में उपयोग के लिए चरों की उनकी प्रति होगी। यदि इन चरों पर कोई संशोधन किया जाता है, तो केवल वह उदाहरण प्रभावित होगा, और अन्य सभी वर्ग उदाहरण अप्रभावित रहते हैं।

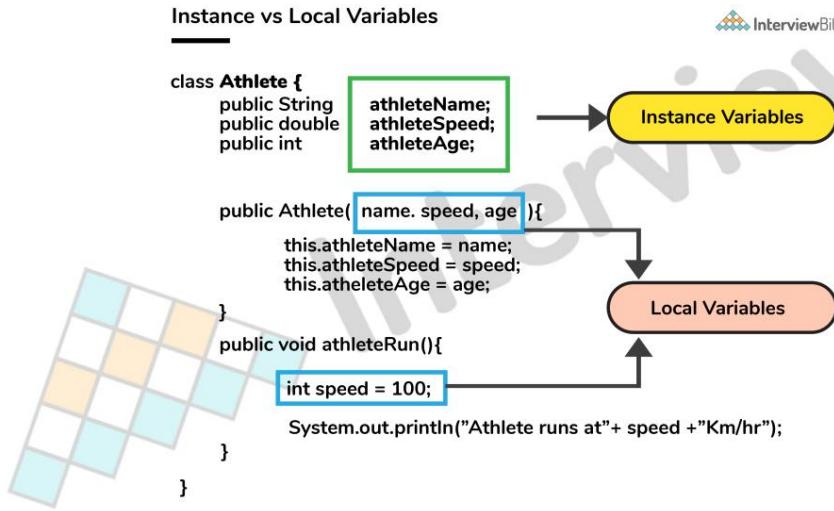
उदाहरण:

```
क्लास एथलीट { सार्वजनिक
स्ट्रिंग एथलीटनाम; सार्वजनिक डबल एथलीट
स्पीड; पब्लिक इंट एथलीटएज; }
```

स्थानीय चर वे चर हैं जो किसी ब्लॉक, फ़ंक्शन या कंस्ट्रक्टर के भीतर मौजूद होते हैं और केवल उनके अंदर ही पहुँचा जा सकता है। चर का उपयोग ब्लॉक दायरे तक ही सीमित है। जब भी किसी विधि के अंदर एक स्थानीय चर घोषित किया जाता है, तो अन्य वर्ग विधियों को स्थानीय चर के बारे में कोई जानकारी नहीं होती है।

उदाहरण:

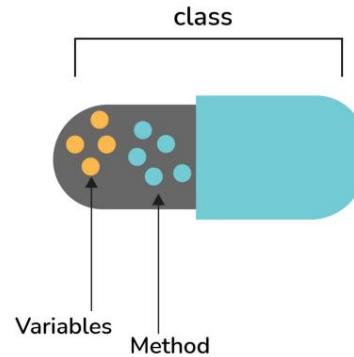
```
सार्वजनिक शून्य एथलीट () {
    स्ट्रिंग एथलीटनाम; डबल एथलीट
    स्पीड; इंट एथलीटएज; }
```



5. डेटा एनकैप्सुलेशन से आप क्या समझते हैं?

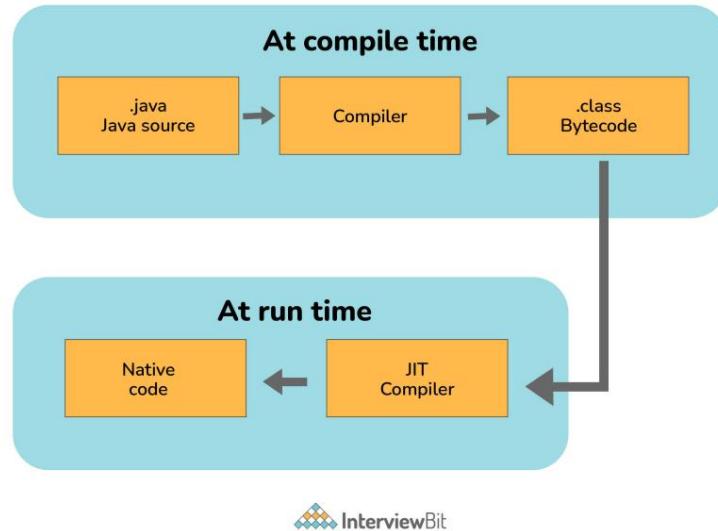
- डेटा एनकैप्सुलेशन एक ऑब्जेक्ट-ओरिएंटेड प्रोग्रामिंग है डेटा विशेषताओं और उनके व्यवहार को एक इकाई में छिपाने की अवधारणा।
- यह डेवलपर्स को यह सुनिश्चित करके सॉवर विकसित करते समय प्रतिरूपकरण का पालन करने में मदद करता है कि प्रत्येक वस्तु अपनी विधियों, विशेषताओं और कार्यात्मकताओं के द्वारा अन्य वस्तुओं से स्वतंत्र है।
- इसका उपयोग किसी वस्तु के निजी गुणों की सुरक्षा के लिए किया जाता है और इसलिए डेटा छिपाने के उद्देश्य से कार्य करता है।

```
class
{
    data members (properties)
    +
    methods (behavior)
}
```



6. हमें जेआईटी कंपाइलर के बारे में कुछ बताएं।

- JIT का मतलब जस्ट-इन-टाइम है और इसका उपयोग रन टाइम के दौरान प्रदर्शन में सुधार के लिए किया जाता है। यह एक ही समय में समान कार्यक्षमता वाले बाइट कोड के कुछ हिस्सों को संकलित करने का कार्य करता है जिससे कोड को चलाने के लिए संकलन समय की मात्रा कम हो जाती है।
- कंपाइलर और कुछ नहीं बल्कि मशीन-निष्पादन योग्य कोड के स्रोत कोड का अनुवादक है। लेकिन जेआईटी कंपाइलर के बारे में क्या खास है? आइए देखें कि यह कैसे काम करता है:
 - सबसे पहले, जावा सोर्स कोड (.java) को बाइट कोड (.class) में कनवर्ट करना javac कंपाइलर की मदद से होता है।
 - फिर, जेवीएम द्वारा .class फाइलें रन टाइम पर लोड की जाती हैं और एक दुभाषिया की मदद से इन्हें मशीन समझने योग्य कोड में बदल दिया जाता है।
 - जेआईटी कंपाइलर जेवीएम का एक हिस्सा है। जब जेआईटी कंपाइलर सक्षम होता है, तो जेवीएम .class फ़ाइलों में विधि कॉल का विश्लेषण करता है और उन्हें अधिक कुशल और मूल कोड प्राप्त करने के लिए संकलित करता है। यह यह भी सुनिश्चित करता है कि प्राथमिकता वाली विधि कॉल अनुकूलित हैं।
 - एक बार उपरोक्त चरण पूरा हो जाने के बाद, JVM कोड को फिर से व्याख्या करने के बजाय सीधे अनुकूलित कोड निष्पादित करता है। यह निष्पादन के प्रदर्शन और गति को बढ़ाता है।



7. क्या आप बराबर () विधि और के बीच अंतर बता सकते हैं जावा में समानता ऑपरेटर (==)?

बराबर ()	==
यह ऑब्जेक्ट क्लास में परिभाषित एक विधि है।	यह जावा में एक बाइनरी ऑपरेटर है।
इस पद्धति का उपयोग निर्दिष्ट व्यावसायिक तर्क के अनुसार दो वस्तुओं के बीच सामग्री की समानता की जाँच के लिए किया जाता है।	इस ऑपरेटर का उपयोग पते (या संदर्भ) की तुलना करने के लिए किया जाता है, अर्थात् यह जांचता है कि क्या दोनों ऑब्जेक्ट एक ही मेमोरी लोकेशन की ओर इशारा कर रहे हैं।

टिप्पणी:

- ऐसे मामलों में जहां किसी वर्ग में बराबर विधि को ओवरराइड नहीं किया जाता है, तब वर्ग समान विधि के डिफ़ॉल्ट कार्यान्वयन का उपयोग करता है जो मूल वर्ग के सबसे नज़दीक है।
- ऑब्जेक्ट क्लास को सभी जावा क्लासेस का पैरेंट क्लास माना जाता है। ऑब्जेक्ट क्लास में बराबर विधि का कार्यान्वयन दो वस्तुओं की तुलना करने के लिए == ऑपरेटर का उपयोग करता है। इस डिफ़ॉल्ट कार्यान्वयन को व्यावसायिक तर्क के अनुसार ओवरराइड किया जा सकता है।

8. जावा में अनंत लूप कैसे घोषित किया जाता है?

अनंत लूप वे लूप हैं जो बिना किसी टूटने की स्थिति के अनंत रूप से चलते हैं।

होशपूर्वक अनंत लूप घोषित करने के कुछ उदाहरण हैं:

- लूप के लिए उपयोग करना:

```
के लिए (;;) {
```

```
// व्यापार का तर्क
// कोई भी ब्रेक लॉजिक
}
```

- जबकि लूप का उपयोग करना:

```
जबकि (सच) {
```

```
// व्यापार का तर्क
// कोई भी ब्रेक लॉजिक
}
```

- करते समय लूप का उपयोग करना:

```
करना{
```

```
// व्यापार का तर्क
// कोई भी ब्रेक लॉजिक }
जबकि (सच);
```

9. कंस्ट्रक्टर ओवरलोडिंग की अवधारणा को संक्षेप में समझाएं

कंस्ट्रक्टर ओवरलोडिंग, कंस्ट्रक्टर मापदंडों में अंतर के साथ एक ही नाम के वर्ग में कई कंस्ट्रक्टर बनाने की प्रक्रिया है।

पैरामीटरों की संख्या और उनके संगत प्रकारों के आधार पर, कंपाइलर द्वारा विभिन्न प्रकार के कंस्ट्रक्टरों की पहचान की जाती है।

```
क्लास हॉस्पिटल { इंट
    वेरिएबल1, वेरिएबल2; दोहरा चर3; सार्वजनिक
    अस्पताल (इंट डॉक्टर, इंट नर्स) {variable1 =
        डॉक्टर; चर 2 = नर्स; } सार्वजनिक अस्पताल (इंट डॉक्टर) { वेरिएबल1 = डॉक्टर; }
    सार्वजनिक अस्पताल(दोगुना वेतन) {variable3 = वेतन} }
```



Constructor Overloading

```
class Hospital {
    int variable1, variable2;
    double variable3;

    public Hospital(int doctors, int nurses) {
        variable1 = doctors;
        variable2 = nurses;
    }

    public Hospital(int doctors) {
        variable1 = doctors;
    }

    public Hospital(double salaries) {
        variable3 = salaries
    }
}
```

3 constructors
overloaded with
different
parameters

यहां तीन कंस्ट्रक्टर परिभाषित किए गए हैं लेकिन वे पैरामीटर प्रकार और उनकी संख्या के आधार पर भिन्न हैं।

10. Comment on method overloading and overriding by citing relevant examples.

In Java, **method overloading** is made possible by introducing different methods in the same class consisting of the same name. Still, all the functions differ in the number or type of parameters. It takes place inside a class and enhances program readability.

The only difference in the return type of the method does not promote method overloading. The following example will furnish you with a clear picture of it.

```
class OverloadingHelp {
    public int findarea (int l, int b) {
        int var1;
        var1 = l * b;
        return var1;
    }
    public int findarea (int l, int b, int h) {
        int var2;
        var2 = l * b * h;
        return var2;
    }
}
```

Method Overloading

```
class OverloadingHelp {
```

```
    public int findarea (int l, int b) {
```

```
        int var1;
        var1 = l * b;
        return var1
    }
```

Same method
name but different
parameters

```
    public int findarea (int l, int b, int h) {
```

```
        int var2;
        var2 = l * b * h;
        return var2;
    }
}
```

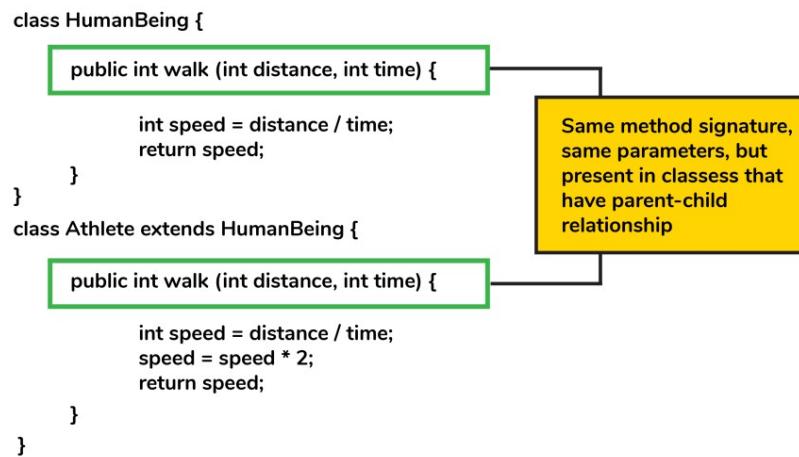
Both the functions have the same name but differ in the number of arguments. The first method calculates the area of the rectangle, whereas the second method calculates the area of a cuboid.

Method overriding is the concept in which two methods having the same method signature are present in two different classes in which an inheritance relationship is present. A particular method implementation (already present in the base class) is possible for the derived class by using method overriding.

Let's give a look at this example:

```
class HumanBeing {
    public int walk (int distance, int time) {
        int speed = distance / time;
        return speed;
    }
}
class Athlete extends HumanBeing {
    public int walk(int distance, int time) {
        int speed = distance / time;
        speed = speed * 2;
        return speed;
    }
}
```

Method Overriding



दोनों वर्ग विधियों का नाम चलना और समान पैरामीटर, दूरी और समय है। यदि व्युत्पन्न वर्ग विधि को कहा जाता है, तो व्युत्पन्न वर्ग के आधार वर्ग विधि को ओवरराइड कर दिया जाता है।

11. जावा प्रोग्राम में सिंगल ट्राई ब्लॉक और मल्टीपल कैच ब्लॉक एक साथ मौजूद हो सकते हैं। समझाना।

हाँ, कई कैच ब्लॉक मौजूद हो सकते हैं लेकिन विशिष्ट दृष्टिकोण सामान्य दृष्टिकोण से पहले आना चाहिए क्योंकि कैच की स्थिति को संतुष्ट करने वाला केवल पहला कैच ब्लॉक ही निष्पादित होता है। दिया गया कोड उसी को दर्शाता है:

```
पब्लिक क्लास मल्टीपलकैच { सार्वजनिक स्थैतिक
शून्य मुख्य (स्ट्रिंग तर्क [])] { कोशिश { int n = 1000, x = 0; इंट एआर []
= नया इंट [एन]; के लिए (int i = 0; i <= n; i++) { arr[i] = i / x; } }
पकड़े ( ArrayIndexOutOfBoundsException अपवाद)
{ System.out.println ("पहला ब्लॉक =
ArrayIndexOutOfBoundsException"); } पकड़े (अंकगणित
अपवाद अपवाद) { System.out.println ("दूसरा ब्लॉक =
अंकगणित अपवाद"); } पकड़ (अपवाद अपवाद) { System.out.println
("तीसरा ब्लॉक = अपवाद"); } } }
```

यहाँ, दूसरा कैच ब्लॉक $0 (i / x)$ से विभाजित होने के कारण निष्पादित किया जाएगा। यदि $x \neq 0$ से अधिक था तो पहला कैच ब्लॉक निष्पादित होगा क्योंकि लूप के लिए $i = n$ तक चलता है और सरणी अनुक्रमणिका $n-1$ तक है।

12. वेरिएबल, मेथड और में फाइनल कीवर्ड के उपयोग की व्याख्या करें कक्षा।

जावा में, अंतिम कीवर्ड का उपयोग किसी चीज़ को स्थिर / अंतिम के रूप में परिभाषित करने के लिए किया जाता है और गैर-पहुंच संशोधक का प्रतिनिधित्व करता है।

- अंतिम चर:

- जब जावा में एक चर को अंतिम घोषित किया जाता है, तो मान को असाइन किए जाने के बाद उसे संशोधित नहीं किया जा सकता है।
- यदि उस वेरिएबल को कोई मान नहीं दिया गया है, तो उसे केवल क्लास के कंस्ट्रक्टर द्वारा ही असाइन किया जा सकता है। अंतिम विधि:
-
- अंतिम घोषित की गई विधि को उसके बच्चों की कक्षाओं द्वारा ओवरराइड नहीं किया जा सकता है।
- एक कंस्ट्रक्टर को अंतिम के रूप में चिह्नित नहीं किया जा सकता है क्योंकि जब भी कोई वर्ग विरासत में मिलता है, तो कंस्ट्रक्टर विरासत में नहीं मिलते हैं। इसलिए, इसे अंतिम रूप देने का कोई मतलब नहीं है। जावा यह कहते हुए संकलन त्रुटि फेंकता है - संशोधक अंतिम यहाँ अनुगति नहीं है

- अंतिम कक्षा:

- अंतिम घोषित वर्ग से कोई वर्ग विरासत में नहीं लिया जा सकता है। लेकिन वह अंतिम वर्ग इसके उपयोग के लिए अन्य वर्गों का विस्तार कर सकता है।

13. क्या अंतिम, अंतिम और अंतिम कीवर्ड का कार्य समान है?

प्रोग्रामिंग करते समय तीनों कीवर्ड की अपनी-अपनी उपयोगिता होती है।

अंतिम: यदि कक्षाओं, चर या विधियों के लिए किसी प्रतिबंध की आवश्यकता होती है, तो अंतिम कीवर्ड काम में आता है।

अंतिम वर्ग की विरासत और अंतिम विधि को ओवरराइड करना अंतिम कीवर्ड के उपयोग से प्रतिबंधित है। अंतिम कीवर्ड को शामिल करते हुए परिवर्तनीय मान स्थिर हो जाता है। उदाहरण:

```
अंतिम इंट ए = 100; ए = 0; // गलती
```

दूसरा कथन एक त्रुटि फेंक देगा।

अंत में: यह एक प्रोग्राम में मौजूद ब्लॉक है जहां इसके अंदर लिखे गए सभी कोड अपवादों को संभालने के बावजूद निष्पादित होते हैं। उदाहरण:

कोशिश

```

करें { int चर = 5; } पकड़
(अपवाद अपवाद)
{ System.out.println ("अपवाद हुआ"); } अंत में
{ System.out.println (" आखिरकार ब्लॉक का निष्पादन"); }

```

अंतिम रूप दें: किसी वस्तु के कचरा संग्रह से पहले, अंतिम विधि को कहा जाता है ताकि सफाई गतिविधि को लागू किया जा सके। उदाहरण:

```

सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग [] तर्क) {स्ट्रिंग उदाहरण = नया स्ट्रिंग
("साक्षात्कार बिट"); उदाहरण = शून्य; सिस्टम.जीसी () // कचरा संग्रहकर्ता कहा जाता
है } सार्वजनिक शून्य को अंतिम रूप दें () {
// अंतिम रूप कहा जाता है }

```

14. आप सुपर कीवर्ड का उपयोग कब कर सकते हैं?

- सुपर कीवर्ड का उपयोग छिपे हुए क्षेत्रों और मूल वर्ग के ओवरराइड विधियों या विशेषताओं तक पहुंचने के लिए किया जाता है।
- निम्नलिखित मामले हैं जब इस कीवर्ड का उपयोग किया जा सकता है: माता-पिता
 - वर्ग के डेटा सदस्यों तक पहुंचना जब वर्ग और उसके बच्चे उपवर्गों के सदस्य नाम समान होते हैं।
 - चाइल्ड क्लास के अंदर पैरेंट क्लास के डिफॉल्ट और पैरामीटराइज्ड कंस्ट्रक्टर को कॉल करने के लिए।
 - जब चाइल्ड क्लास ने उन्हें ओवरराइड किया हो, तो पैरेंट क्लास के तरीकों को एक्सेस करना।
- निम्न उदाहरण सभी 3 मामलों को प्रदर्शित करता है जब एक सुपर कीवर्ड का उपयोग किया जाता है।

```

पब्लिक क्लास पेरेंट{
    निजी पूर्णांक संख्या = 1;

    माता-पिता ()
        {System.out.println ("मूल वर्ग डिफॉल्ट निर्माता।");
    }

    पेरेंट (स्ट्रिंग x)
        {System.out.println ("पेरेंट क्लास पैरामीटराइज्ड कंस्ट्रक्टर।");
    }

    सार्वजनिक शून्य फू ()
        {System.out.println ("मूल वर्ग फू!");
    }
}

पब्लिक क्लास चाइल्ड माता-पिता का विस्तार करता है { निजी
int num = 2;

चाइल्ड ()
    { System.out.println ("चाइल्ड क्लास डिफॉल्ट कंस्ट्रक्टर");

    उत्तम(); // डिफॉल्ट पेरेंट कंस्ट्रक्टर सुपर ("कॉल पेरेंट") को कॉल करने के लिए;
                // पैरामीटराइज्ड कंस्ट्रक्टर को कॉल करने के लिए।
}

शून्य प्रिंटनग ()
    { System.out.println (संख्या);
        System.out.println (super.num); // मूल वर्ग की संख्या का मान प्रिंट करता है
    }

@Override
public void foo()
    { System.out.println ("पेरेंट क्लास फू!"); सुपर.फू (); // ओवरराइड
        फू के अंदर पेरेंट क्लास की फू मेथड को कॉल करें
    }
}

```

15. क्या स्थैतिक विधियों को अतिभारित किया जा सकता है?

हाँ! एक ही नाम वाली कक्षा में दो या दो से अधिक स्थिर विधियाँ हो सकती हैं, लेकिन अलग-अलग इनपुट पैरामीटर।

16. क्या स्थैतिक विधियों को ओवरराइड किया जा सकता है?

- नहीं! उपर्युक्त में समान हस्ताक्षर वाले स्थैतिक तरीकों की घोषणा की जा सकती है लेकिन ऐसे मामलों में रन टाइम पॉलीमॉर्फिज्म नहीं हो सकता है।
- ओवरराइडिंग या डायनेमिक पॉलीमॉर्फिज्म रनटाइम के दौरान होता है, लेकिन स्टैटिक मेथड्स को लोड किया जाता है और स्टैटिकली कंपाइल टाइम पर देखा जाता है। इसलिए, इन विधियों को ओवरराइड नहीं किया जा सकता है।

17. कचरा संग्रहण का मुख्य उद्देश्य क्या है?

इस प्रक्रिया का मुख्य उद्देश्य जावा प्रोग्राम के निष्पादन के दौरान उन अगम्य वस्तुओं को हटाकर अनावश्यक और अगम्य वस्तुओं द्वारा कब्जा की गई मेमोरी स्पेस को मुक्त करना है।

- यह सुनिश्चित करता है कि स्मृति संसाधन कुशलता से उपयोग किया जाता है, लेकिन यह कोई गारंटी नहीं देता है कि प्रोग्राम निष्पादन के लिए पर्याप्त स्मृति होगी।

18. मेमोरी का कौन सा हिस्सा - स्टैक या हीप - कचरे में साफ किया जाता है संग्रह प्रक्रिया?

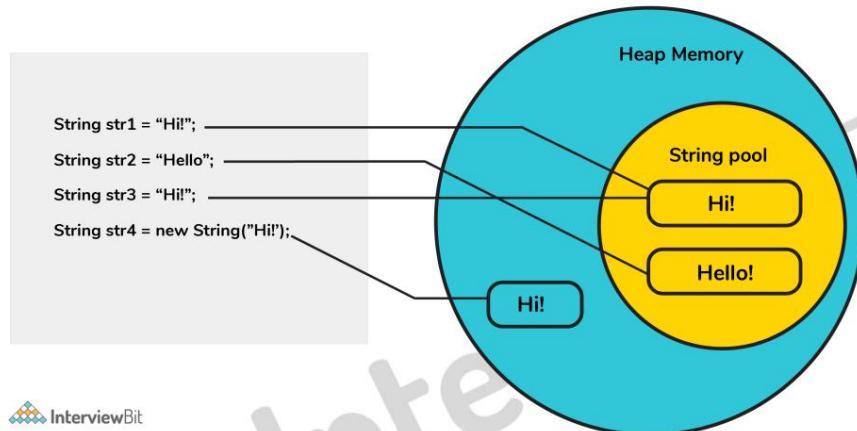
ढेर।

जावा इंटरमीडिएट साक्षात्कार प्रश्न

19. सुरक्षा पहलू के अलावा इसके पीछे क्या कारण हैं? जावा में तार अपरिवर्तनीय बनाना?

निम्नलिखित कारणों से एक स्ट्रिंग को अपरिवर्तनीय बनाया जाता है:

- स्ट्रिंग पूल: जावा के डिजाइनर इस तथ्य से अवगत थे कि स्ट्रिंग डेटा प्रकार का उपयोग प्रोग्रामर और डेवलपर्स द्वारा प्रमुख रूप से किया जा रहा है। इस प्रकार, वे शुरू से ही अनुकूलन चाहते थे। वे स्ट्रिंग अक्षर को संग्रहीत करने के लिए स्ट्रिंग पूल (जावा हीप में एक भंडारण क्षेत्र) का उपयोग करने की धारणा के साथ आए। उनका इरादा साझाकरण की मदद से अस्थायी स्ट्रिंग ऑब्जेक्ट को कम करना था। साझा करने की सुविधा के लिए एक अपरिवर्तनीय वर्ग की आवश्यकता है। दो अज्ञात पक्षों के बीच परिवर्तनशील संरचनाओं का बंटवारा संभव नहीं है। इस प्रकार, अपरिवर्तनीय जावा स्ट्रिंग स्ट्रिंग पूल की अवधारणा को क्रियान्वित करने में मदद करता है।

String Pool

- **मल्टीथ्रेडिंग:** स्ट्रिंग ऑब्जेक्ट्स के संबंध में थ्रेड्स की सुरक्षा जावा में एक महत्वपूर्ण पहलू है। यदि स्ट्रिंग ऑब्जेक्ट अपरिवर्तनीय हैं, तो किसी बाहरी सिंक्रनाइज़ेशन की आवश्यकता नहीं है। इस प्रकार, स्ट्रिंग ऑब्जेक्ट्स को विभिन्न थ्रेड्स में साझा करने के लिए एक क्लीनर कोड लिखा जा सकता है। संगामिति की जटिल प्रक्रिया को इस विधि द्वारा सुगम बनाया जाता है।
- **संग्रह:** हैशटेबल्स और हैश मैप्स के मामले में, कुंजी स्ट्रिंग ऑब्जेक्ट हैं। यदि स्ट्रिंग ऑब्जेक्ट अपरिवर्तनीय नहीं हैं, तो इसे उस अवधि के दौरान संशोधित किया जा सकता है जब यह हैश मैप्स में रहता है। नतीजतन, वांछित डेटा की पुनर्प्राप्ति संभव नहीं है। ऐसे बदलते राज्य बहुत जोखिम पैदा करते हैं। इसलिए, स्ट्रिंग को अपरिवर्तनीय बनाना काफी सुरक्षित है।

20. आप String, StringBuffer, के बीच अंतर कैसे करेंगे?

और एक स्ट्रिंगबिल्डर?

- भंडारण क्षेत्र: स्ट्रिंग में, स्ट्रिंग पूल भंडारण क्षेत्र के रूप में कार्य करता है। StringBuilder और StringBuffer के लिए, हीप मेमोरी भंडारण क्षेत्र है।
- उत्परिवर्तनीयता: एक स्ट्रिंग अपरिवर्तनीय है, जबकि स्ट्रिंगबिल्डर और स्ट्रिंगबफर दोनों उत्परिवर्तनीय हैं।
- दक्षता: स्ट्रिंग के साथ काम करना काफी धीमा है। हालाँकि, StringBuilder संचालन करने में सबसे तेज़ है। StringBuffer की गति String से अधिक और StringBuilder से कम होती है। (उदाहरण के लिए स्ट्रिंगबिल्डर में एक चरित्र जोड़ना सबसे तेज़ है और स्ट्रिंग में बहुत धीमा है क्योंकि नए स्ट्रिंग के लिए संलग्न चरित्र के साथ एक नई मेमोरी की आवश्यकता होती है।)
- थ्रेड-सुरक्षित: थ्रेडेड वातावरण के मामले में, स्ट्रिंगबिल्डर और स्ट्रिंगबफर का उपयोग किया जाता है जबकि स्ट्रिंग का उपयोग नहीं किया जाता है। हालाँकि, StringBuilder एकल थ्रेड वाले वातावरण के लिए उपयुक्त है, और StringBuffer कई थ्रेड्स के लिए उपयुक्त है।

वाक्य - विन्यास:

```
// डोरी
स्ट्रिंग पहले = "साक्षात्कार बिट";
स्ट्रिंग दूसरा = नया स्ट्रिंग ("साक्षात्कार बिट");
// स्ट्रिंगबफर
स्ट्रिंगबफर तीसरा = नया स्ट्रिंगबफर ("इंटरव्यूबिट");
// स्ट्रिंगबिल्डर
स्ट्रिंगबिल्डर चौथा = नया स्ट्रिंगबिल्डर ("इंटरव्यूबिट");
```

21. प्रासंगिक गुणों का उपयोग करना के बीच के अंतरों को उजागर करता है इंटरफेस और अमूर्त वर्ग।

- विधियों की उपलब्धता: इंटरफ़ेस में केवल अमूर्त विधियाँ उपलब्ध हैं, जबकि अमूर्त वर्गों में अमूर्त विधियों के साथ गैर-अमूर्त विधियाँ मौजूद हो सकती हैं।
- चर प्रकार: स्थैतिक और अंतिम चर केवल इंटरफ़ेस के मामले में घोषित किए जा सकते हैं, जबकि अमूर्त वर्गों में गैर-स्थैतिक और गैर-अंतिम चर भी हो सकते हैं।
- इनहेरिटेंस: इंटरफ़ेस द्वारा मल्टीपल इनहेरिटेंस की सुविधा होती है, जबकि एब्स्ट्रैक्ट क्लासेस मल्टीपल इनहेरिटेंस को बढ़ावा नहीं देते हैं।
- डेटा सदस्य अभिगम्यता: डिफ़ॉल्ट रूप से, इंटरफ़ेस के वर्ग डेटा सदस्य सार्वजनिक प्रकार के होते हैं। इसके विपरीत, एक अमूर्त वर्ग के वर्ग के सदस्यों को संरक्षित या निजी भी किया जा सकता है।
- कार्यान्वयन: एक अमूर्त वर्ग की मदद से, एक इंटरफ़ेस का कार्यान्वयन आसानी से संभव है। हालाँकि, इसका विलोम सत्य नहीं है; सार वर्ग उदाहरण:

सार्वजनिक अमूर्त वर्ग एथलीट { सार्वजनिक अमूर्त शून्य चलना (); }

इंटरफ़ेस उदाहरण:

सार्वजनिक इंटरफ़ेस चलने योग्य { शून्य चलना (); }

22. जावा में, स्थिर और साथ ही निजी पद्धति को ओवरराइड करना संभव है। कथन पर टिप्पणी करें।

संदर्भ में दिया गया यह कथन पूर्णतः असत्य है। स्थैतिक विधियों का वस्तुओं के साथ कोई संबंध नहीं है, और ये विधियाँ वर्ग स्तर की हैं। चाइल्ड क्लास के मामले में, पैरेंट क्लास की तरह ही मेथड सिग्नेचर के साथ स्टैटिक मेथड बिना किसी कंपाइलेशन एरर को फेंके भी मौजूद हो सकता है।

यहां वर्णित घटना को लोकप्रिय रूप से छिपाने की विधि के रूप में जाना जाता है, और ओवरराइडिंग निश्चित रूप से संभव नहीं है। निजी पद्धति को ओवरराइड करना अकल्पनीय है क्योंकि निजी पद्धति की दृश्यता केवल मूल वर्ग तक ही सीमित है। नतीजतन, केवल छिपाने की सुविधा दी जा सकती है और ओवरराइडिंग नहीं।

23. क्या हैशसेट को ट्रीसेट से अलग बनाता है?

हालांकि हैशसेट और ट्रीसेट दोनों सिंक्रनाइज़ नहीं हैं और यह सुनिश्चित करते हैं कि डुप्लिकेट मौजूद नहीं हैं, कुछ ऐसे गुण हैं जो हैशसेट को ट्रीसेट से अलग करते हैं।

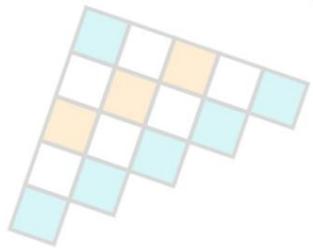
- कार्यान्वयन:** हैशसेट के लिए, हैश तालिका का उपयोग तत्वों को अनियंत्रित तरीके से संग्रहीत करने के लिए किया जाता है। हालांकि, ट्रीसेट तत्वों को क्रमबद्ध तरीके से संग्रहीत करने के लिए लाल-काले पेड़ का उपयोग करता है।
- जटिलता/प्रदर्शन:** तत्वों को जोड़ने, पुनर्प्राप्त करने और हटाने के लिए, समय परिशोधित जटिलता हैशसेट के लिए ओ (1) है। समान संचालन करने के लिए समय जटिलता ट्रीसेट के लिए थोड़ी अधिक है और ओ (लॉग एन) के बराबर है। कुल मिलाकर, हैशसेट का प्रदर्शन ट्रीसेट की तुलना में तेज है।
- विधियाँ:** हैशकोड () और बराबर () वस्तुओं के बीच तुलना करने के लिए हैशसेट द्वारा उपयोग की जाने वाली विधियाँ हैं। इसके विपरीत, तुलना करने के लिए () और तुलना () विधियों का उपयोग ट्रीसेट द्वारा वस्तु तुलना की सुविधा के लिए किया जाता है।
- ऑब्जेक्ट प्रकार:** विषम और अशक्त वस्तुओं को हैशसेट की मदद से संग्रहीत किया जा सकता है। ट्रीसेट के मामले में, विषम वस्तुओं या अशक्त वस्तुओं को सम्मिलित करते समय रनटाइम अपवाद होता है।

24. भंडारण के लिए स्ट्रिंग पर वर्ण सरणी को प्राथमिकता क्यों दी जाती है गोपनीय जानकारी?

जावा में, एक स्ट्रिंग मूल रूप से अपरिवर्तनीय है यानी इसे संशोधित नहीं किया जा सकता है। अपनी घोषणा के अनुसार, यह तब तक स्ट्रिंग पूल में बना रहता है जब तक इसे कचरे के रूप में नहीं हटाया जाता है। दूसरे शब्दों में, एक स्ट्रिंग एक अनियमित और अनिर्दिष्ट समय अंतराल के लिए मेमोरी के हीप सेक्शन में रहती है एयर स्ट्रिंग वैल्यू प्रोसेसिंग निष्पादित की जाती है।

नतीजतन, हैकर्स द्वारा हानिकारक गतिविधियों को आगे बढ़ाने के लिए महत्वपूर्ण जानकारी चुराई जा सकती है यदि उनके द्वारा मेमोरी डंप को अवैध रूप से एक्सेस किया जाता है। इस तरह के जोखिमों को किसी भी चर को संग्रहीत करने के लिए परिवर्तनशील वस्तुओं या चरित्र सरणियों जैसी संरचनाओं का उपयोग करके समाप्त किया जा सकता है। एर वर्ण सरणी चर का कार्य किया जाता है, चर को उसी पल में रिक्त करने के लिए कॉन्फ़िगर किया जा सकता है। नतीजतन, यह हीप मेमोरी को बचाने में मदद करता है और हैकर्स को महत्वपूर्ण डेटा निकालने का कोई मौका नहीं देता है।

25. जावा में JVM, JRE और JDK में क्या अंतर हैं?



मानदंड	जेडीके	जेआरई	जेवीएम
संक्षेपाक्षर	जावा विकास किट	जावा क्रम पर्यावरण	जावा वर्चुअल मशीन
परिभाषा	JDK एक पूर्ण है सोवेयर जावा अनुप्रयोगों के विकास के लिए विकास किट। इसमें JRE, JavaDoc, कंपाइलर, डिबगर्स, आदि।	जेआरई एक है सोवेयर पैकेज प्रदान करना जावा क्लास लाइब्रेरी, JVM और सभी आवश्यक घटकों को चलाने के लिए जावा अनुप्रयोग।	जेवीएम एक प्लेटफॉर्म पर निर्भर, अमूर्त मशीन है जिसमें विनिर्देश शामिल हैं - दस्तावेज़ का वर्णन करना जेवीएम कार्यान्वयन आवश्यकताएं, कंप्यूटर प्रोग्राम जेवी आवश्यकताओं को पूरा करता है और निष्पादन के लिए इंस्टेंस ऑब्जेक्ट
मुख्य उद्देश्य	JDK मुख्य रूप से कोड विकास और निष्पादन के लिए उपयोग किया जाता है।	जेआरई मुख्य रूप से के लिए प्रयोग किया जाता है d . को निष्पादित करने के लिए पर्यावरण निर्माण	जेवीएम जेआरई को सभी कार्यान्वयन के लिए विनिर्देश प्रदान करता है।

26. हैश मैप और हैशटेबल में क्या अंतर है जावा में?

हैश मैप	हैश टेबल
हैश मैप सिंक्रनाइज़ नहीं है जिससे गैर-थ्रेडेड अनुप्रयोगों के लिए इसे बेहतर बना दिया गया है।	हैशटेबल सिंक्रनाइज़ है और इसलिए यह थ्रेडेड अनुप्रयोगों के लिए उपयुक्त है।
केवल एक नल कुंजी की अनुमति देता है लेकिन मूल्यों में किसी भी संख्या में शून्य।	यह दोनों कुंजियों या मानों में शून्य की अनुमति नहीं देता है।
अपने उपर्युक्त LinkedHashMap का उपयोग करके सम्मिलन के आदेश का समर्थन करता है।	हैशटेबल में प्रविष्टि के आदेश की गारंटी नहीं है।

27. जावा में प्रतिबिंब का क्या महत्व है?

- रिफ्लेक्शन शब्द का प्रयोग किसी कोड की स्वयं या उसके सिस्टम पर किसी कोड की निरीक्षण क्षमता का वर्णन करने के लिए किया जाता है और इसे रनटाइम के दौरान संशोधित किया जाता है।
- एक उदाहरण पर विचार करें जहां हमारे पास अज्ञात प्रकार की वस्तु है और हमारे पास एक विधि 'fooBar ()' है जिसे हमें ऑब्जेक्ट पर कॉल करने की आवश्यकता है। जावा का स्टैटिक टाइपिंग सिस्टम इस मेथड इनवोकेशन की अनुमति नहीं देता है जब तक कि ऑब्जेक्ट के प्रकार को पहले से नहीं जाना जाता है। यह प्रतिबिंब का उपयोग करके प्राप्त किया जा सकता है जो कोड को ऑब्जेक्ट को स्कैन करने और यह पहचानने की अनुमति देता है कि क्या इसमें "fooBar ()" नामक कोई विधि है और केवल तभी विधि को कॉल करें यदि आवश्यक हो।

```
विधि विधिऑफफू = fooObject.getClass ()।getMethod ("fooBar", शून्य); methodOfFoo.invoke (fooObject,
शून्य);
```

- प्रतिबिंब का उपयोग करने के अपने स्वयं के विपक्ष
 - हैं: गति - प्रतिबिंब के कारण विधि आमंत्रण प्रत्यक्ष विधि कॉल की तुलना में लगभग तीन गुना धीमा है।
 - सुरक्षा टाइप करें - जब प्रतिबिंब का उपयोग करके गलत तरीके से इसके संदर्भ के माध्यम से एक विधि लागू की जाती है, तो आमंत्रण रनटाइम पर विफल हो जाता है क्योंकि यह संकलन/लोड समय पर नहीं पाया जाता है।
 - ट्रेसिबिलिटी - जब भी कोई चिंतनशील विधि विफल हो जाती है, तो एक विशाल स्टैक ट्रेस के कारण इस विफलता के मूल कारण को खोजना बहुत मुश्किल होता है। किसी को मूल कारण की पहचान करने के लिए इनवोक () और प्रॉक्सी () विधि लॉग में गहराई से जाना होगा।
- इसलिए, उन समाधानों का पालन करने की सलाह दी जाती है जिनमें प्रतिबिंब शामिल नहीं है और अंतिम उपाय के रूप में इस पद्धति का उपयोग करें।

28. धागे के उपयोग के विभिन्न तरीके क्या हैं?

- हम जावा में एक थ्रेड को दो तरीकों से परिभाषित और कार्यान्वित कर सकते हैं:
 - थ्रेड क्लास का विस्तार

```
क्लास इंटरव्यूबिट थ्रेड उदाहरण थ्रेड बढ़ाता है { सार्वजनिक शून्य रन () {System.out.println
("थ्रेड रन ...");
```

```
} सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग तर्क []) {साक्षात्कार बिट थ्रेड
उदाहरण आईबी = नया साक्षात्कार बिट थ्रेड उदाहरण (); आईबी.स्टार्ट ();
```

```
}
```

- रननेबल इंटरफ़ेस को लागू करना

```
क्लास इंटरव्यूबिट थ्रेड उदाहरण रननेबल लागू करता है { सार्वजनिक शून्य रन () {System.out.println
("थ्रेड रन ...");
```

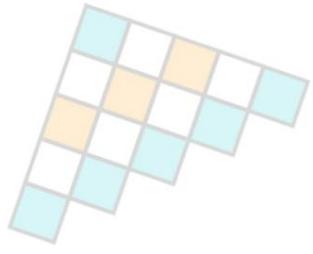
```
} सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग आर्म []) {
थ्रेड आईबी = नया थ्रेड (नया साक्षात्कार बिट थ्रेड उदाहरण ()); आईबी.स्टार्ट ();
```

```
}
```

- रननेबल इंटरफ़ेस की विधि का उपयोग करके थ्रेड को कार्यान्वित करना अधिक पसंदीदा और लाभप्रद है क्योंकि जावा में कक्षाओं के एकाधिक विरासत के लिए समर्थन नहीं है।
- प्रारंभ () विधि का उपयोग थ्रेड निष्पादन के लिए एक अलग कॉल स्टैक बनाने के लिए किया जाता है। कॉल स्टैक बनने के बाद, JVM उस कॉल स्टैक में थ्रेड को निष्पादित करने के लिए रन () विधि को कॉल करता है।

29. कंस्ट्रक्टर और मेथड में क्या अंतर हैं

जावा में एक वर्ग का?



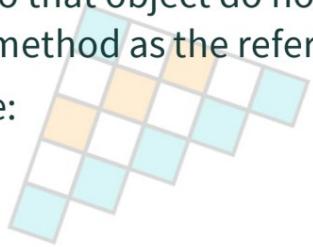
निर्माता	तरीका
<p>ऑब्जेक्ट स्टेट को इनिशियलाइज़ करने के लिए कंस्ट्रक्टर का उपयोग किया जाता है।</p>	<p>वस्तु के व्यवहार को उजागर करने के लिए विधि का उपयोग किया जाता है।</p>
<p>कंस्ट्रक्टर का कोई रिटर्न प्रकार नहीं है।</p>	<p>विधि में वापसी प्रकार होना चाहिए। यहां तक कि अगर यह कुछ भी वापस नहीं करता है, तो वापसी प्रकार शून्य है।</p>
<p>कंस्ट्रक्टर परोक्ष रूप से आह्वान किया जाता है।</p>	<p>वस्तु पर विधि को स्पष्ट रूप से लागू किया जाना है।</p>
<p>यदि कंस्ट्रक्टर परिभाषित नहीं है, तो जावा कंपाइलर द्वारा एक डिफॉल्ट कंस्ट्रक्टर प्रदान किया जाता है।</p>	<p>यदि कोई विधि परिभाषित नहीं है, तो संकलक इसे प्रदान नहीं करता है।</p>
<p>कंस्ट्रक्टर का नाम वर्ग के नाम के बराबर होना चाहिए।</p>	<p>विधि के नाम का कोई भी नाम हो सकता है या वर्ग का नाम भी हो सकता है।</p>
<p>एक कंस्ट्रक्टर को अंतिम के रूप में चिह्नित नहीं किया जा सकता है क्योंकि जब भी कोई वर्ग विरासत में मिलता है, तो कंस्ट्रक्टर विरासत में नहीं मिलते हैं।</p> <p>इसलिए, इसे अंतिम रूप देने का कोई मतलब नहीं है। जावा यह कहते हुए संकलन त्रुटि फेंकता है - संशोधक अंतिम की अनुमति नहीं है</p>	<p>एक विधि को अंतिम के रूप में परिभाषित किया जा सकता है लेकिन इसके उपवर्गों में इसे ओवरराइड नहीं किया जा सकता है।</p>

30. Java works as “pass by value” or “pass by reference” phenomenon?

Java always works as a “pass by value”. There is nothing called a “pass by reference” in Java. However, when the object is passed in any method, the address of the value is passed due to the nature of object handling in Java. When an object is passed, a copy of the reference is created by Java and that is passed to the method. The objects point to the same memory location. 2 cases might happen inside the method:

- **Case 1:** When the object is pointed to another location: In this case, the changes made to that object do not get reflected the original object before it was passed to the method as the reference points to another location.

For example:



```
कक्षा साक्षात्कार बिटटेस्ट { इंट संख्या;
```

```
    इंटरव्यूबिटटेस्ट (इंट एक्स) {संख्या = एक्स;
```

```
}
```

```
    इंटरव्यूबिटटेस्ट () {संख्या = 0;
```

```
}
```

```
} क्लास ड्राइवर { सार्वजनिक
```

```
    स्थैतिक शून्य मुख्य (स्ट्रिंग [] args) {
```

```
        // एक संदर्भ बनाएं इंटरव्यूबिटटेस्ट
```

```
        ibTestObj = नया इंटरव्यूबिटटेस्ट (20); // अपडेटऑब्जेक्ट मेथड अपडेटऑब्जेक्ट (ibTestObj) का  
        संदर्भ पास करें; // अपडेटऑब्जेक्ट निष्पादित होने के बाद, ऑब्जेक्ट में num के मान की जांच करें।
```

```
        System.out.println (ibTestObj.num);
```

```
} सार्वजनिक स्थैतिक शून्य अद्यतन वस्तु (साक्षात्कार बिटटेस्ट ibObj) {
```

```
        // ऑब्जेक्ट को नए संदर्भ में इंगित करें ibObj = नया इंटरव्यूबिटटेस्ट  
(()); // ibObj.num = 50 का मान अपडेट करें;
```

```
}
```

```
}
```

```
आउटपुट:
```

```
20
```

- केस 2: जब ऑब्जेक्ट रेफरेंस को संशोधित नहीं किया जाता है: इस मामले में, चूंकि हमारे पास एक ही मेमोरी लोकेशन की ओर इशारा करते हुए संदर्भ की कॉपी है, ऑब्जेक्ट की सामग्री में कोई भी बदलाव मूल ऑब्जेक्ट में परिलक्षित होता है।

उदाहरण के लिए:

कक्षा साक्षात्कार बिटटेस्ट { इंट संख्या;

इंटरव्यूबिटटेस्ट (इंट एक्स) {संख्या = एक्स;

}

इंटरव्यूबिटटेस्ट () {संख्या = 0;

}

} क्लास ड्राइवर

{ सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग [] args) {

// एक संदर्भ बनाएं इंटरव्यूबिटटेस्ट

ibTestObj = नया इंटरव्यूबिटटेस्ट (20); // अपडेटऑब्जेक्ट मेथड अपडेटऑब्जेक्ट (ibTestObj) का संदर्भ पास करें; // अपडेटऑब्जेक्ट निष्पादित होने के बाद, ऑब्जेक्ट में num के मान की जांच करें।

System.out.println (ibTestObj.num);

} सार्वजनिक स्थैतिक शून्य अद्यतन वस्तु (साक्षात्कार बिटटेस्ट ibObj) {

// ibObj को नए स्थान पर इंगित करने के लिए कोई परिवर्तन नहीं किया गया है // num का मान अपडेट करें
ibObj.num = 50;

}

}

आउटपुट:
50

31. डेटा में बहुत सारे अपडेट किए जाने की आवश्यकता होने पर स्ट्रिंग या स्ट्रिंग बफर में से किसे प्राथमिकता दी जानी चाहिए?

StringBuffer प्रकृति में परिवर्तनशील और गतिशील है जबकि String अपरिवर्तनीय है। स्ट्रिंग का प्रत्येक अद्यतन/संशोधन एक नई स्ट्रिंग बनाता है जिससे अनावश्यक वस्तुओं के साथ स्ट्रिंग पूल ओवरलोड हो जाता है। इसलिए, बहुत सारे अपडेट के मामलों में, हमेशा स्ट्रिंगबफर का उपयोग करना पसंद किया जाता है क्योंकि यह स्ट्रिंग पूल में कई स्ट्रिंग ऑब्जेक्ट्स के निर्माण के ओवरहेड को कम कर देगा।

32. किसी वर्ग की विशेषताओं के क्रमांकन की अनुमति कैसे न दें
जावा?

- इसे प्राप्त करने के लिए, विशेषता को क्षणिक कीवर्ड के उपयोग के साथ घोषित किया जा सकता है जैसा कि नीचे दिखाया गया है:

```
पब्लिक क्लास इंटरव्यूबिटउदाहरण {
    निजी क्षणिक स्ट्रिंग someInfo; निजी स्ट्रिंग नाम; निजी इंट आईडी; // :
    // गेटर्स सेटर्स //:
}
```

- उपरोक्त उदाहरण में, someInfo को छोड़कर सभी क्षेत्रों को क्रमबद्ध किया जा सकता है।

33. क्या होता है यदि स्थैतिक संशोधक को शामिल नहीं किया जाता है जावा में मुख्य विधि हस्ताक्षर?

कोई संकलन त्रुटि नहीं होगी। लेकिन फिर प्रोग्राम चलाया जाता है, क्योंकि JVM मुख्य विधि हस्ताक्षर को मैप नहीं कर सकता है, कोड रनटाइम पर "NoSuchMethodError" त्रुटि फेंकता है।

34. क्या होता है यदि एक के अंदर कई मुख्य विधियां हैं जावा में कक्षा?

कार्यक्रम संकलित नहीं हो सकता क्योंकि संकलक कहता है कि विधि को कक्षा के अंदर पहले ही परिभाषित किया जा चुका है।

35. ऑब्जेक्ट क्लोनिंग से आप क्या समझते हैं और आप कैसे करते हैं? जावा में इसे प्राप्त करें?

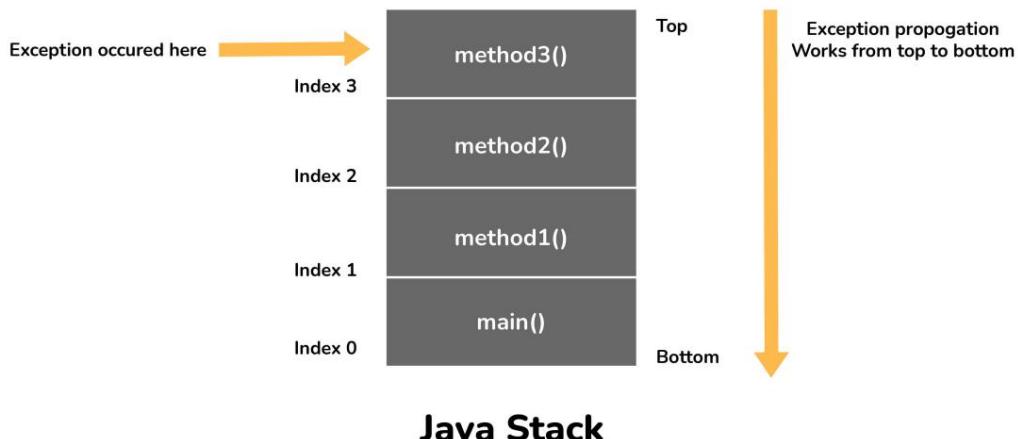
- यह किसी भी वस्तु की स्टीक प्रतिलिपि बनाने की प्रक्रिया है। इसका समर्थन करने के लिए, एक जावा वर्ग को java.lang पैकेज के क्लोनेबल इंटरफ़ेस को लागू करना होगा और ऑब्जेक्ट क्लास द्वारा प्रदान की गई क्लोन () विधि को ओवरराइड करना होगा, जिसका सिंटैक्स है:

```
संरक्षित वस्तु क्लोन () CloneNotSupportedException फेंकता है { रिटर्न (ऑब्जेक्ट) super.clone (); }
```

- यदि क्लोन करने योग्य इंटरफ़ेस लागू नहीं किया गया है और केवल विधि को ओवरराइड किया गया है, तो इसका परिणाम Java में CloneNotSupportedException होता है।

36. कोड में अपवाद कैसे फैलता है?

जब कोई अपवाद होता है, तो सबसे पहले यह मेल खाने वाले कैच ब्लॉक का पता लगाने के लिए खोज करता है। यदि मैचिंग कैच ब्लॉक स्थित है, तो उस ब्लॉक को निष्पादित किया जाएगा। अन्यथा, अपवाद विधि कॉल स्टैक के माध्यम से फैलता है और कॉलर विधि में जाता है जहां कैच ब्लॉक के मिलान की प्रक्रिया की जाती है। यह प्रसार तब तक होता है जब तक कि मैचिंग कैच ब्लॉक नहीं मिल जाता। यदि मैच नहीं मिलता है, तो कार्यक्रम मुख्य विधि में समाप्त हो जाता है।



37. क्या एक कोशिश के दौरान कैच ब्लॉक का पालन करना अनिवार्य है? खंड मैथा?

नहीं, कैच ब्लॉक के लिए ट्राइ ब्लॉक में उपस्थित होना आवश्यक नहीं है। - एक कोशिश ब्लॉक का पालन कैच ब्लॉक या अंत में ब्लॉक द्वारा किया जाना चाहिए। यदि अपवादों की संभावना अधिक है, तो उन्हें विधि के श्रो क्लॉज का उपयोग करके घोषित किया जाना चाहिए।

38. क्या रिटर्न के समय अंत में ब्लॉक निष्पादित हो जाएगा?

कोशिश ब्लॉक और कैच ब्लॉक के अंत में कथन लिखा गया है जैसा कि नीचे दिखाया गया है?

```

सार्वजनिक int someMethod (int i) { कोशिश
करें { // कुछ स्टेटमेंट रिटर्न 1; }कैच(अपवाद ई)
{ // कुछ स्टेटमेंट रिटर्न 999; }
आखिरकार{ //आखिरकार बयानों को
ब्लॉक करें

}
}
  
```

अंत में ब्लॉक को अपवाद के बावजूद निष्पादित किया जाएगा या नहीं। एकमात्र मामला जहां अंत में ब्लॉक निष्पादित नहीं किया जाता है, जब उसे कोशिश/पकड़ ब्लॉक में कहीं भी 'System.exit ()' विधि का सामना करना पड़ता है।

39. क्या आप किसी क्लास के कंस्ट्रक्टर को दूसरे कंस्ट्रक्टर के अंदर बुला सकते हैं?

हाँ, अवधारणा को कंस्ट्रक्टर चेनिंग कहा जा सकता है और इसका उपयोग करके प्राप्त किया जा सकता है
यह ()।

```

public class InterviewBit{           Constructor Chaining
    InterviewBit(){
        this("Hi InterviewBit!");
    }
    InterviewBit(String s){
        System.out.println(s);
    }
    public static void main(String[] args){
        InterviewBit ib = new InterviewBit();
    }
}

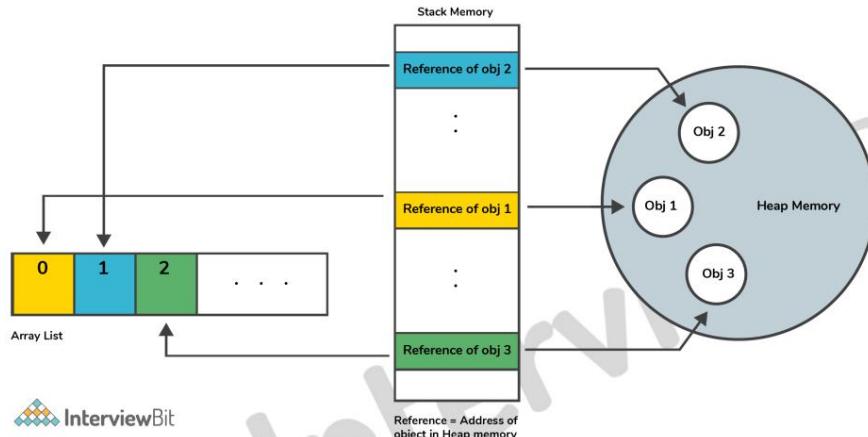
```



40. सन्निहित स्मृति स्थानों का उपयोग आमतौर पर किसी सरणी में वास्तविक मानों को संग्रहीत करने के लिए किया जाता है, लेकिन ArrayList में नहीं। समझाना।

ArrayList के मामले में, आदिम डेटा प्रकारों (जैसे int, float, आदि) के रूप में डेटा संग्रहीत करना संभव नहीं है। ArrayList में मौजूद डेटा सदस्यों/ऑब्जेक्ट्स में उन ऑब्जेक्ट्स के संदर्भ होते हैं जो मेमोरी में विभिन्न साइटों पर स्थित होते हैं। इस प्रकार, वास्तविक वस्तुओं या गैर-आदिम डेटा प्रकारों (जैसे पूर्णांक, डबल, आदि) का भंडारण विभिन्न स्मृति स्थानों में होता है।

Sorting of objects in ArrayList

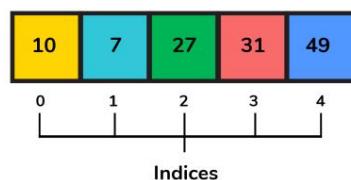


हालाँकि, यह सरणियों पर लागू नहीं होता है। ऑब्जेक्ट या आदिम प्रकार के मानों को सन्निहित स्मृति स्थानों में सरणियों में संग्रहीत किया जा सकता है, इसलिए प्रत्येक तत्व को अगले तत्व के संदर्भ की आवश्यकता नहीं होती है।

Sorting values in Array

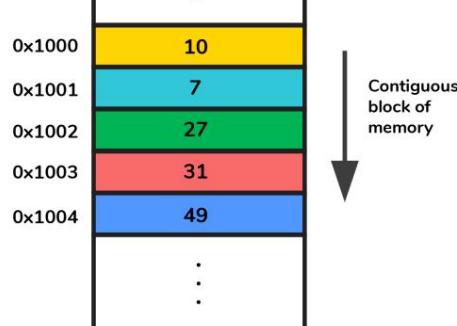
How Array look like?

Array



InterviewBit

Array in memory



जावा उन्नत साक्षात्कार प्रश्न

41. हालांकि वंशानुक्रम एक लोकप्रिय औरोपी अवधारणा है, यह संरचना से कम फायदेमंद है। समझाना।

निम्नलिखित परिदृश्यों में वंशानुक्रम रचना से पिछड़ जाता है:

- जावा में बहु-विरासत संभव नहीं है। कक्षाएं केवल एक सुपरक्लास से विस्तारित हो सकती हैं। ऐसे मामलों में जहां कई कार्यात्मकताओं की आवश्यकता होती है, उदाहरण के लिए - फ़ाइल में जानकारी को पढ़ने और लिखने के लिए, संरचना के पैटर्न को प्राथमिकता दी जाती है। लेखक के साथ-साथ पाठक कार्यात्मकताओं को निजी सदस्यों के रूप में मान कर उनका उपयोग किया जा सकता है।
- संरचना उच्च लचीलापन प्राप्त करने में सहायता करती है और इनकैप्सुलेशन को तोड़ने से रोकती है।
- रचना के साथ इकाई परीक्षण संभव है न कि वंशानुक्रम के साथ। जब कोई डेवलपर एक अलग वर्ग की रचना करने वाले वर्ग का परीक्षण करना चाहता है, तो परीक्षण की सुविधा के लिए रचित वर्ग को इंगित करने के लिए मॉक ऑब्जेक्ट बनाया जा सकता है। वंशानुक्रम की सहायता से यह तकनीक संभव नहीं है क्योंकि वंशानुक्रम में सुपरक्लास की सहायता के बिना व्युत्पन्न वर्ग का परीक्षण नहीं किया जा सकता है।
- संरचना की शिथिल युग्मित प्रकृति विरासत की कसकर युग्मित प्रकृति की तुलना में बेहतर है।

आइए एक उदाहरण लेते हैं:

```
पैकेज तुलना; पब्लिक क्लास टॉप
{ पब्लिक इंट स्टार्ट () { रिटर्न 0;
} } क्लास बॉटम एक्सटेंड टॉप { पब्लिक इंट स्टॉप
() { रिटर्न 0; } }
```

उपरोक्त उदाहरण में, वंशानुक्रम का अनुसरण किया जाता है। अब, शीर्ष वर्ग में कुछ संशोधन इस प्रकार किए गए हैं:

```
पब्लिक क्लास टॉप { पब्लिक इंटर्फ़ेस
    स्टार्ट () { रिटर्न 0; } सार्वजनिक शून्य
    रोक () { } }
```

यदि शीर्ष वर्ग के नए कार्यान्वयन का पालन किया जाता है, तो नीचे की कक्षा में एक संकलन-समय त्रुटि उत्पन्न होना तय है। `Top.stop()` फ़ंक्शन के लिए असंगत वापसी प्रकार है। संगतता सुनिश्चित करने के लिए शीर्ष या निचले वर्ग में परिवर्तन करना होगा। हालाँकि, दी गई समस्या को हल करने के लिए रचना तकनीक का उपयोग किया जा सकता है:

```
कक्षा नीचे {शीर्ष बराबर =
    नया शीर्ष (); सार्वजनिक इंटर्फ़ेस () {
        पर.start (); पार.स्टॉप (); वापसी
        0; } }
```

42. नए () का उपयोग करके एक स्ट्रिंग का निर्माण एक शाब्दिक से अलग कैसे है?

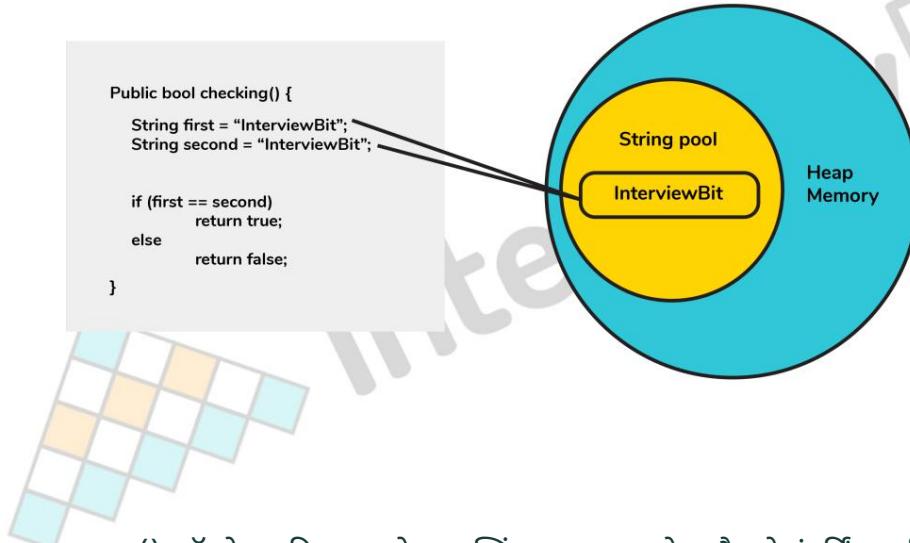
जब एक स्ट्रिंग को असाइनमेंट ऑपरेटर की सहायता से एक शाब्दिक के रूप में बनाया जाता है, तो यह स्ट्रिंग निरंतर पूल में अपना रास्ता बनाता है ताकि स्ट्रिंग इंटर्निंग हो सके।

यदि सामग्री दोनों के लिए समान है, तो ढेर में समान वस्तु को एक अलग स्ट्रिंग द्वारा संदर्भित किया जाएगा।

```
सार्वजनिक बूल जाँच () {
    स्ट्रिंग पहले = "साक्षात्कार बिट";
    स्ट्रिंग सेकंड = "साक्षात्कार बिट"; अगर (पहला == दूसरा)
    सच लौटाता है; अन्यथा झूठी वापसी; }
```

चेकिंग () फ़ंक्शन सत्य वापस आ जाएगा क्योंकि एक ही सामग्री को दोनों चर द्वारा संदर्भित किया जाता है।

String Pool by means of assignment operator



इसके विपरीत, जब एक नए () ऑपरेटर की मदद से एक स्ट्रिंग का गठन होता है, तो इंटर्निंग नहीं होती है। ऑब्जेक्ट हीप मेमोरी में बन जाता है, भले ही वही कंटेंट ऑब्जेक्ट मौजूद हो।

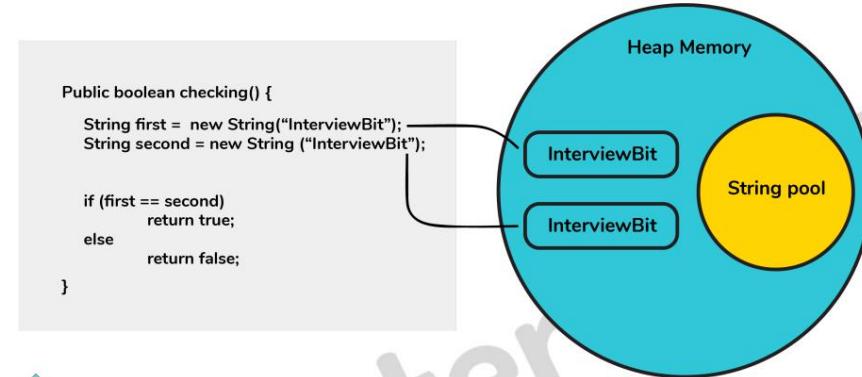
```

सार्वजनिक बूल जाँच () {
    स्ट्रिंग पहले = नया स्ट्रिंग ("साक्षात्कार बिट");
    स्ट्रिंग दूसरा = नया स्ट्रिंग ("साक्षात्कार बिट"); अगर (पहला == दूसरा) सच लौटाता
    है; अन्यथा झूठी वापसी; }

```

चेकिंग () फ़ंक्शन झूठी वापसी करेगा क्योंकि एक ही सामग्री दोनों चर द्वारा संदर्भित नहीं है।

String Pool by means of new operator



43. क्या कचरा संग्रहकर्ता होने के बावजूद किसी प्रोग्राम में स्मृति सीमा को पार करना संभव है?

हाँ, कचरा संग्रहकर्ता की उपस्थिति के बावजूद कार्यक्रम के लिए स्मृति से बाहर जाना संभव है। कचरा संग्रह उन वस्तुओं को पहचानने और नष्ट करने में सहायता करता है जिनकी अब कार्यक्रम में आवश्यकता नहीं है, ताकि उनके द्वारा उपयोग किए जाने वाले संसाधनों को मुक्त किया जा सके।

एक प्रोग्राम में, यदि कोई वस्तु पहुंच योग्य नहीं है, तो उस वस्तु के संबंध में कचरा संग्रहण का निष्पादन होता है। यदि एक नई वस्तु बनाने के लिए आवश्यक मेमोरी की मात्रा पर्याप्त नहीं है, तो उन वस्तुओं के लिए मेमोरी जारी की जाती है जो अब कचरा संग्रहकर्ता की मदद से दायरे में नहीं हैं। जब जारी की गई मेमोरी नई वस्तुओं को बनाने के लिए पर्याप्त नहीं होती है तो प्रोग्राम के लिए स्मृति सीमा पार हो जाती है।

इसके अलावा, ढेर मेमोरी की थकावट तब होती है जब वस्तुओं को इस तरह से बनाया जाता है कि वे दायरे में रहें और स्मृति का उपभोग करें। डेवलपर को यह सुनिश्चित करना चाहिए कि जिस वस्तु का काम पूरा हो गया है, उसे हटाना है। हालांकि कचरा संग्रहकर्ता जितना संभव हो सके स्मृति को पुनः प्राप्त करने के लिए अपने स्तर पर सर्वोत्तम प्रयास करता है, फिर भी स्मृति सीमा को पार किया जा सकता है।

आइए निम्नलिखित उदाहरण पर एक नज़र डालें:

```
सूची <स्ट्रिंग> उदाहरण = नई लिंकडिलिस्ट <स्ट्रिंग> (); जबकि (सत्य) { example.add (नई  
स्ट्रिंग ("मेमोरी लिमिट पार हो गई")); }
```

44. सिंक्रनाइज़ेशन क्यों आवश्यक है? a . की सहायता से स्पष्ट कीजिए प्रासंगिक उदाहरण।

विभिन्न प्रक्रियाओं के समर्ती निष्पादन को सिंक्रनाइज़ेशन द्वारा संभव बनाया गया है।

जब एक विशेष संसाधन को कई थ्रेड्स के बीच साझा किया जाता है, तो ऐसी स्थितियाँ उत्पन्न हो सकती हैं जिनमें कई थ्रेड्स को समान साझा संसाधन की आवश्यकता होती है।

सिंक्रनाइज़ेशन समस्या को हल करने में सहायता करता है और संसाधन एक समय में एक थ्रेड द्वारा साझा किया जाता है। आइए इसे और अधिक स्पष्ट रूप से समझने के लिए एक उदाहरण लेते हैं। उदाहरण के लिए, आपके पास एक URL है और आपको इसके लिए किए गए अनुरोधों की संख्या का पता लगाना है। एक साथ दो अनुरोध गिनती को अनिश्चित बना सकते हैं।

कोई सिंक्रनाइज़ेशन नहीं:

```
पैकेज गुमनाम; पब्लिक क्लास  
काउंटिंग {  
    निजी इंट वृद्धि_काउंटर ; सार्वजनिक अंतर वृद्धि ()  
    { वृद्धि_काउंटर = वृद्धि_काउंटर + 1; वापसी वृद्धि_काउंटर;  
  
    }  
}
```

Without Synchronization

```
package anonymous;
public class Counting {
    private int increase_counter;
    public int increase() {
        increase_counter = increase_counter + 1;
        return increase_counter;
    }
}
```

①

When Thread T1 comes, increase_counter value becomes 11, if increase_counter was 10 before.

Shared Resource

②

Simultaneously, when Thread T2 comes, the value of increase_counter is viewed as 10 incorrectly instead of 11

 InterviewBit

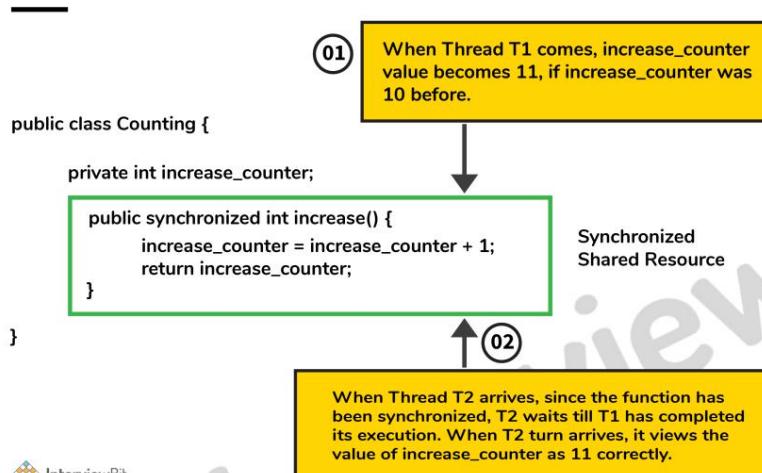
यदि कोई थ्रेड 1 गिनती को 10 के रूप में देखता है, तो इसे 1 से 11 तक बढ़ा दिया जाएगा।

इसके साथ ही, यदि कोई अन्य थ्रेड 2 गिनती को 10 के रूप में देखता है, तो इसे 1 से 11 तक बढ़ा दिया जाएगा। इस प्रकार, गणना मानों में असंगति होती है क्योंकि अपेक्षित अंतिम मान 12 है लेकिन वास्तविक अंतिम मान जो हमें मिलेगा वह 11 होगा।

अब, फंक्शन वृद्धि () को सिंक्रनाइज़ किया गया है ताकि एक साथ एक्सेसिंग न हो सके।

सिंक्रनाइज़ेशन के साथ:

```
पैकेज गुमनाम; पब्लिक क्लास
काउंटिंग {
    निजी int वृद्धि_काउंटर ; सार्वजनिक सिंक्रनाइज़ इंट
    वृद्धि () { वृद्धि_काउंटर = वृद्धि_काउंटर + 1; वापसी वृद्धि_काउंटर;
    }
}
```

With Synchronization

यदि कोई थ्रेड थ्रेड1 गिनती को 10 के रूप में देखता है, तो इसे 1 से 11 तक बढ़ाया जाएगा, फिर थ्रेड थ्रेड 2 गिनती को 11 के रूप में देखेगा, इसे 1 से 12 तक बढ़ाया जाएगा। इस प्रकार, गणना मानों में स्थिरता होती है।

45. नीचे दिए गए कूट में... का क्या महत्व है?

```

सार्वजनिक शून्य fooBarMethod (स्ट्रिंग ... चर) {
    // विधि कोड
}

```

- प्रदान करने की क्षमता ... एक विशेषता है जिसे varargs (चर तर्क) कहा जाता है जिसे जावा 5 के भाग के रूप में पेश किया गया था।
- उपरोक्त उदाहरण में ... वाले फ़ंक्शन इंगित करता है कि यह डेटाटाइप स्ट्रिंग के कई तर्क प्राप्त कर सकता है।
- उदाहरण के लिए, fooBarMethod को कई तरीकों से बुलाया जा सकता है और हमारे पास डेटा को संसाधित करने के लिए अभी भी एक विधि हो सकती है जैसा कि नीचे दिखाया गया है:

```

fooBarMethod ("फू", "बार");
fooBarMethod ("फू", "बार", "बू"); fooBarMethod
(नया स्ट्रिंग [] {"foo", "var", "boo"}); सार्वजनिक शून्य myMethod (स्ट्रिंग ... चर)
{ के लिए (स्ट्रिंग चर: चर) { // व्यापार तर्क

}
}

```

46. क्या आप जावा थ्रेड जीवनचक्र की व्याख्या कर सकते हैं?

जावा थ्रेड जीवन चक्र इस प्रकार है:

- नया - जब थ्रेड का उदाहरण बनाया जाता है और प्रारंभ () विधि को लागू नहीं किया जाता है, तो थ्रेड को जीवित माना जाता है और इसलिए नई स्थिति में।
- रननेबल - एक बार स्टार्ट () विधि लागू होने के बाद, जेवीएम द्वारा रन () विधि को कॉल करने से पहले, थ्रेड को रननेबल (चलाने के लिए तैयार) स्थिति में कहा जाता है। इस अवस्था को थ्रेड के वेटिंग या स्लीपिंग स्टेट से भी प्रविष्ट किया जा सकता है।
- रनिंग - जब रन () विधि लागू की जाती है और थ्रेड अपना निष्पादन शुरू करता है, तो थ्रेड को रनिंग स्थिति में कहा जाता है।
- नॉन-रननेबल (ब्लॉकड / वेटिंग) - जब थ्रेड अपने जीवंत होने के तथ्य के बावजूद चलने में सक्षम नहीं होता है, तो थ्रेड को नॉन-रननेबल अवस्था में कहा जाता है।

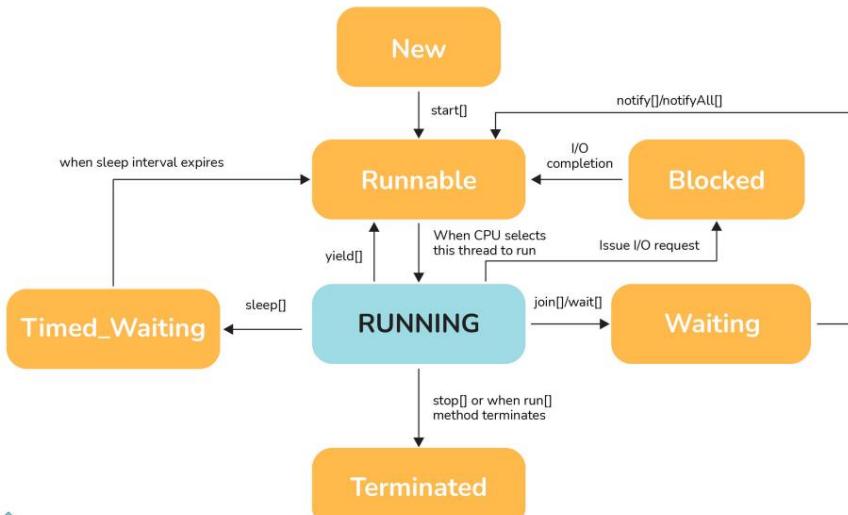
आदर्श रूप से, अपनी जीवंतता के कुछ समय के लिए, धागा एक चलने योग्य स्थिति में जाना चाहिए।

○ एक थ्रेड को अवरुद्ध स्थिति में कहा जाता है यदि वह सिंक्रनाइज़ कोड दर्ज करना चाहता है लेकिन यह असमर्थ है क्योंकि उसी ऑफेक्ट पर उस सिंक्रनाइज़ ब्लॉक में एक और थ्रेड चल रहा है। पहले थ्रेड को तब तक इंतजार करना पड़ता है जब तक कि दूसरा थ्रेड सिंक्रोनाइज़ ब्लॉक से बाहर न निकल जाए।

○ एक थ्रेड को वेटिंग अवस्था में कहा जाता है यदि वह किसी अन्य थ्रेड से सिग्नल के निष्पादित होने की प्रतीक्षा कर रहा है, अर्थात् यह सिग्नल प्राप्त होने तक काम की प्रतीक्षा करता है।

- समाप्त - एक बार रन () विधि निष्पादन पूरा हो जाने के बाद, थ्रेड को टर्मिनेटेड चरण में प्रवेश करने के लिए कहा जाता है और इसे जीवित नहीं माना जाता है।

निम्नलिखित फ्लोचार्ट स्पष्ट रूप से जावा में धागे के जीवनचक्र की व्याख्या करता है।



47. an . के उपयोग के बीच क्या समझौता हो सकता है
ऑर्डर किए गए सरणी के उपयोग बनाम अनियंत्रित सरणी?

- ऑर्डर किए गए सरणी होने का मुख्य लाभ ओ (लॉग एन) की कम खोज समय जटिलता है जबकि एक अनियंत्रित सरणी में समय जटिलता है औ (एन)।
- आदेशित सरणी का मुख्य दोष इसका बढ़ा हुआ सम्मिलन समय है जो इस तथ्य के कारण $O(n)$ है कि इसके तत्व को प्रत्येक प्रविष्टि के दौरान सरणी के क्रम को बनाए रखने के लिए पुनः व्यवस्थित करना पड़ता है जबकि अनियंत्रित सरणी में समय जटिलता केवल $O(1)$ है।
- उपरोक्त 2 प्रमुख बिंदुओं को ध्यान में रखते हुए और एक डेवलपर को किस तरह के परिदृश्य की आवश्यकता है, इसके आधार पर कार्यान्वयन के लिए उपयुक्त डेटा संरचना का उपयोग किया जा सकता है।

48. क्या जावा में एक ही वर्ग या पैकेज को दो बार आयात करना संभव है और रनटाइम के दौरान इसका क्या होता है?

एक वर्ग या पैकेज को एक से अधिक बार आयात करना संभव है, हालांकि, यह बेमानी है क्योंकि JVM आंतरिक रूप से पैकेज या वर्ग को केवल एक बार लोड करता है।

49. यदि किसी पैकेज में उप पैकेज हैं, तो क्या यह केवल मुख्य पैकेज का आयात करने के लिए पर्याप्त होगा? उदाहरण के लिए क्या com.myMainPackage.* का आयात भी com.myMainPackage.mySubPackage.* आयात करता है?

यह एक बड़ा NO है। हमें यह समझने की ज़रूरत है कि पैकेज के उप-पैकेजों का आयात स्पष्ट रूप से किया जाना चाहिए। मूल पैकेज को आयात करने से केवल उसके भीतर की कक्षाओं का आयात होता है, न कि उसके बच्चे/उप-पैकेजों की सामग्री का।

50. यदि कोड System.exit(0) हो तो क्या अंत में ब्लॉक निष्पादित किया जाएगा try ब्लॉक के अंत में लिखा होता है?

ना। प्रोग्राम पोस्ट का नियंत्रण System.exit(0) तुरंत चला जाता है और प्रोग्राम समाप्त हो जाता है, यही कारण है कि अंत में ब्लॉक कभी भी निष्पादित नहीं होता है।

51. जावा में मार्कर इंटरफेस से आप क्या समझते हैं?

मार्कर इंटरफेस, जिन्हें टैगिंग इंटरफेस के रूप में भी जाना जाता है, वे इंटरफेस हैं जिनमें कोई विधियाँ और स्थिरांक परिभाषित नहीं हैं। वे वस्तुओं के संबंध में रन टाइम से संबंधित जानकारी प्राप्त करने के लिए कंपाइलर और जेवीएम की मदद करने के लिए हैं।

52. जावा में "डबल ब्रेस इनिशियलाइज़ेशन" शब्द की व्याख्या करें?

जावा में किसी भी संग्रह को प्रारंभ करने का यह एक सुविधाजनक माध्यम है। नीचे दिए गए उदाहरण पर विचार करें।

```

आयात java.util.HashSet; आयात
java.util.Set;

पब्लिक क्लास IBDoubleBraceDemo { सार्वजनिक
    स्थैतिक शून्य मुख्य (स्ट्रिंग [] args) {
        सेट <स्ट्रिंग> स्ट्रिंगसेट = नया हैशसेट <स्ट्रिंग> () {

            {
                जोड़ ("सेट 1"); जोड़ ("सेट
                2"); जोड़ ("सेट 3");

            }
        };

        कुछ करो (स्ट्रिंगसेट);
    }

    निजी स्थैतिक शून्य कुछ करते हैं (सेट <स्ट्रिंग> स्ट्रिंगसेट) {System.out.println (स्ट्रिंगसेट);

    }
}

```

उपरोक्त उदाहरण में, हम देखते हैं कि स्ट्रिंगसेट को डबल ब्रेसिज़ का उपयोग करके प्रारंभ किया गया था।

- पहला ब्रेस एक अनाम आंतरिक वर्ग बनाने का कार्य करता है जिसमें मूल वर्ग के व्यवहार तक पहुँचने की क्षमता होती है। हमारे उदाहरण में, हम हैशसेट का उपवर्ग बना रहे हैं ताकि वह हैशसेट की ऐड () विधि का उपयोग कर सके।
- दूसरा ब्रेसिज़ इंस्टेंसेस को इनिशियलाइज़ करने का काम करता है।

इस पद्धति के माध्यम से आरंभ करते समय सावधानी बरती जानी चाहिए क्योंकि विधि में अनाम आंतरिक वर्गों का निर्माण शामिल है जो कचरा संग्रह या क्रमांकन प्रक्रियाओं के दौरान समस्या पैदा कर सकता है और इसके परिणामस्वरूप स्मृति रिसाव भी हो सकता है।

53. ऐसा क्यों कहा जाता है कि स्ट्रिंग वर्ग की लंबाई () विधि सटीक परिणाम नहीं देती है?

- लंबाई विधि स्ट्रिंग की यूनिकोड इकाइयों की संख्या लौटाती है। आइए समझते हैं कि यूनिकोड इकाइयाँ क्या हैं और नीचे क्या भ्रम है।
- हम जानते हैं कि जावा स्ट्रिंग प्रतिनिधित्व के लिए UTF-16 का उपयोग करता है। इस यूनिकोड के साथ, हमें नीचे दिए गए दो यूनिकोड संबंधित शब्दों को समझने की आवश्यकता है:
 - कोड बिंदु: यह एक पूर्णांक का प्रतिनिधित्व करता है जो कोड स्थान में एक वर्ण को दर्शाता है।
 - कोड यूनिट: यह कोड बिंदुओं को एन्कोड करने के लिए उपयोग किया जाने वाला एक बिट अनुक्रम है। ऐसा करने के लिए, एक कोड बिंदु का प्रतिनिधित्व करने के लिए एक या अधिक इकाइयों की आवश्यकता हो सकती है।
- UTF-16 योजना के तहत, कोड बिंदुओं को तार्किक रूप से 17 विमानों में विभाजित किया गया था और पहले विमान को मूल बहुभाषी विमान (BMP) कहा जाता था। बीएमपी में क्लासिक वर्ण हैं - यू+0000 से यू+एफएफएफएफ। शेष वर्ण- U+10000 से U+10FFFF को पूरक वर्ण कहा गया क्योंकि वे शेष विमानों में समाहित थे।
 - पहले विमान से कोड बिंदु एक 16-बिट कोड इकाई का उपयोग करके एन्कोड किए गए हैं शेष विमानों से कोड बिंदु दो
 - कोड इकाइयों का उपयोग करके एन्कोड किए गए हैं।

अब यदि एक स्ट्रिंग में पूरक वर्ण होते हैं, तो लंबाई फ़ंक्शन को 2 इकाइयों के रूप में गिना जाएगा और लंबाई () फ़ंक्शन का परिणाम अपेक्षित के अनुसार नहीं होगा।

दूसरे शब्दों में, यदि 2 इकाइयों का 1 पूरक वर्ण है, तो उस एकल वर्ण की लंबाई दो मानी जाती है - यहाँ अशुद्धि पर ध्यान दें? जावा प्रलेखन के अनुसार, यह अपेक्षित है, लेकिन वास्तविक तर्क के अनुसार, यह गलत है।

54. नीचे दिए गए कोड का आउटपुट क्या है और क्यों?

```
पब्लिक क्लास इंटरव्यूबिट { सार्वजनिक स्पैटिक
  शून्य मुख्य (स्ट्रिंग [] तर्क) {
    System.out.println ('बी' + 'आई' + 'टी');
```

```
}
```

यदि अक्षरों का प्रयोग दोहरे उद्धरणों (या स्ट्रिंग अक्षर) में किया गया होता तो "बिट" मुद्रित परिणाम होता। लेकिन प्रश्न में वर्ण अक्षर (एकल उद्धरण) का उपयोग किया जा रहा है, यही कारण है कि संयोजन नहीं होगा। प्रत्येक वर्ण के संबंधित ASCII मान जोड़े जाएंगे और उस योग का परिणाम मुद्रित किया जाएगा।

'बी', 'आई', 'टी' के ASCII मान हैं:

- 'बी' = 98 'आई'
- = 105 'टी' = 116
-

$$98 + 105 + 116 = 319$$

इसलिए 319 मुद्रित किया जाएगा।

55. वस्तु को पात्र बनाने के संभावित तरीके क्या हैं?

जावा में कचरा संग्रह (जीसी)?

पहला दृष्टिकोण: ऑब्जेक्ट निर्माण उद्देश्य पूरा होने के बाद ऑब्जेक्ट संदर्भों को शून्य पर सेट करें।

```
पब्लिक क्लास IBGarbageCollect { सार्वजनिक स्थैतिक
    शून्य मुख्य (स्ट्रिंग [] args){
        स्ट्रिंग s1 = "कुछ स्ट्रिंग"; // s1 स्ट्रिंग ऑब्जेक्ट
        को संदर्भित करता है - अभी तक GC के लिए योग्य नहीं है
        s1 = शून्य; // अब s1 GC के लिए योग्य है
    }
}
```

दूसरा दृष्टिकोण: संदर्भ चर को किसी अन्य वस्तु पर इंगित करें। ऐसा करने से, जिस वस्तु का संदर्भ चर पहले संदर्भित कर रहा था, वह GC के लिए योग्य हो जाती है।

```
पब्लिक क्लास IBGarbageCollect { सार्वजनिक स्थैतिक
    शून्य मुख्य (स्ट्रिंग [] args) {स्ट्रिंग s1 = "कचरा संग्रह करने के लिए"; स्ट्रिंग
        s2 = "एक और वस्तु"; System.out.println(s1); // s1 अभी
        तक GC s1 = s2 के लिए योग्य नहीं है; // s1 को s2 द्वारा इंगित अन्य
        वस्तु की ओर इंगित करें / * यहां, स्ट्रिंग ऑब्जेक्ट जिसमें "टू गारबेज कलेक्ट" सामग्री है, रेफरल नहीं है
    }
}}
```

Third Approach: Island of Isolation Approach: When 2 reference variables pointing to instances of the same class, and these variables refer to only each other and the objects pointed by these 2 variables don't have any other references, then it is said to have formed an “Island of Isolation” and these 2 objects are eligible for GC.

```
public class IBGarbageCollect {  
    IBGarbageCollect ib;  
    public static void main(String [] str){  
        IBGarbageCollect ibgc1 = new IBGarbageCollect();  
        IBGarbageCollect ibgc2 = new IBGarbageCollect();  
        ibgc1.ib = ibgc2; //ibgc1 points to ibgc2  
        ibgc2.ib = ibgc1; //ibgc2 points to ibgc1  
        ibgc1 = null;  
        ibgc2 = null;  
        /*  
         * We see that ibgc1 and ibgc2 objects refer  
         * to only each other and have no valid  
         * references- these 2 objects for island of isolation - eligible for GC  
         */  
    }  
}
```

Java Interview Programs

56. Check if a given string is palindrome using recursion.

```

/*
 * यह जावा प्रोग्राम यह जांचने के लिए कि दी गई इनपुट स्ट्रिंग पैलिंड्रोम है या रिकर्सन का उपयोग नहीं कर रही है */
आयात java.util.*;
पब्लिक क्लास इंटरव्यूबिट
{ सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग args []) {स्कैनर s = नया स्कैनर (System.in); स्ट्रिंग शब्द = s.nextLine (); System.out.println ("क्या "+शब्द+" पैलिंड्रोम है ? - "+isWordPalindrome(word));
}

सार्वजनिक स्थैतिक बूलियन isWordPalindrome (स्ट्रिंग शब्द) {स्ट्रिंग रिवर्सवर्ड =
    getReverseWord (शब्द); // यदि शब्द इसके विपरीत के बराबर है, तो यह एक पैलिंड्रोम
    है if(word.equals(reverseWord)){ return true;

} झूठी वापसी;
}

सार्वजनिक स्थैतिक स्ट्रिंग getReverseWord (स्ट्रिंग शब्द) { अगर (शब्द == शून्य || शब्द।
    isEmpty ()) {
        वापसी शब्द;
    }

    वापसी word.charAt(word.length()- 1) + getReverseWord(word.substring(0, word.len)
}
}
}

```

57. यह जांचने के लिए जावा प्रोग्राम लिखें कि क्या दो तार हैं विपर्यय।

मुख्य विचार तारों की लंबाई को मान्य करना है और फिर यदि समान पाया जाता है, तो स्ट्रिंग को चार सरणी में परिवर्तित करें और फिर सरणी को सॉर्ट करें और जांचें कि दोनों बराबर हैं या नहीं।

```

आयात java.util.Arrays; आयात
java.util.Scanner; पब्लिक क्लास
इंटरव्यूबिट { सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग
[] args) {स्कैनर s = नया स्कैनर (System.in); // दो स्ट्रिंग्स से इनपुट
    System.out.print ("फर्स्ट स्ट्रिंग: "); स्ट्रिंग स्ट्रिंग 1 = s.nextLine
    (); System.out.print ("दूसरा स्ट्रिंग: "); स्ट्रिंग स्ट्रिंग 2 =
    s.nextLine (); // लंबाई की जांच करें if(string1.length() ==
    string2.length ()) { // स्ट्रिंग्स को चार ऐरे में कनवर्ट करें char[]
    characterArray1 = string1.toCharArray (); चार [] वर्णअरे 2
    = string2.toCharArray (); // सरणियों को क्रमबद्ध करें
    Arrays.sort(characterArray1);
    Arrays.sort(characterArray2); // समानता के लिए जाँच करें, यदि समान
    पाया जाता है तो विपर्यय, अन्यथा विपर्यय बूलियन नहीं हैअनाग्राम =
    Arrays.equals(characterArray1, characterArray2); System.out.println
    ("एनाग्राम: " + isAnagram);

}
}

```

58. किसी दी गई संख्या का भाज्य ज्ञात करने के लिए जावा प्रोग्राम लिखें।

```

पब्लिक क्लास फाइंडफैक्टोरियल { सार्वजनिक स्थैतिक
शून्य मुख्य (स्ट्रिंग [] args) { int num = 10; लंबा फैक्टोरियल परिणाम =
1; for(int i = 1; i <= num; ++i) {

    भाज्य परिणाम *= मैं;
}
System.out.println ("फैक्टोरियल: " + फैक्टोरियल रिसेट);
}
}

```

59. 1 से n तक की गैर-डुप्लिकेटिंग संख्याओं की एक सरणी को देखते हुए, जहां एक संख्या गायब है, उस लापता संख्या को खोजने के लिए एक कुशल जावा प्रोग्राम लिखें।

सूत्र का उपयोग करके n प्राकृतिक संख्याओं का योग ज्ञात करना और फिर दिए गए सरणी में संख्याओं का योग ज्ञात करना है। इन दो योगों को घटाने पर वह संख्या प्राप्त होती है जो वास्तविक लुप्त संख्या है। इसका परिणाम ओ (एन) समय जटिलता और ओ (1) अंतरिक्ष जटिलता में होता है।

```
पब्लिक क्लास IBMissingNumberProblem {
    सार्वजनिक स्थेतिक शून्य मुख्य (स्ट्रिंग [] तर्क) {
        इंट [] सरणी = {4,3,8,7,5,2,6}; int लापता संख्या
        = findMissingNum (सरणी); System.out.println ("अनुपलब्ध संख्या
        है " + अनुपलब्ध संख्या);
    }

    सार्वजनिक इंट findMissingNum (int [] सरणी) {
        इंट एन = सरणी।लंबाई + 1; int
        sumOfFirstNNums=n*(n+1)/2; int
        realSumOfArr=0; के लिए (int i = 0; i <
        array.length; i++) {realSumOfArr+=array[i];

        } वापसी sumOfFirstNNums-actualSumOfArr;
    }
}
```

60. यह जांचने के लिए जावा प्रोग्राम लिखें कि कोई संख्या जादुई संख्या है या नहीं। एक संख्या को एक जादुई संख्या कहा जाता है यदि प्रत्येक चरण में अंकों का योग करता है और उस योग के अंकों का योग करता है, तो अंतिम परिणाम (जब केवल एक अंक |e होता है) 1 होता है।

उदाहरण, संख्या पर विचार करें:

- चरण 1: 163 => 1+6+3 = 10
- चरण 2: 10 => 1+0 = 1 => अतः 163 एक जादुई संख्या है

```
पब्लिक क्लास IBMagicNumber{
```

```
    सार्वजनिक स्थैतिक शून्य मुख्य (स्ट्रिंग [] args) { int num = 163; int
        sumOfDigits = 0; जबकि (संख्या > 0 || समआँफडिजिट्स > 9) {

            अगर (संख्या == 0) {

                num = sumOfDigits;
                योगआँफडिजिट्स = 0;

            } sumOfDigits += num% 10; संख्या /
                = 10;
            }

            // यदि योग 1 है, तो मूल संख्या जादुई संख्या है यदि (sumOfDigits == 1)
            {System.out.println ("मैजिक नंबर"); }else { System.out.println (" मैजिक
                नंबर नहीं");

            }

        }
    }
```

निष्कर्ष

61. निष्कर्ष

जावा सरल उच्च-स्तरीय भाषाओं में से एक है जो अनुपयोग विकास के लिए आवश्यक शक्तिशाली उपकरण और प्रभावशाली मानक प्रदान करता है। यह समर्वर्ती-आधारित समस्याओं से निपटने के लिए अद्भुत थ्रेडिंग समर्थन प्रदान करने वाली पहली भाषाओं में से एक थी। उपयोग में आसान सिंटैक्स और जावा की अंतर्निहित विशेषताएं, जो अनुपयोगों को प्रदान की जाने वाली स्थिरता के साथ संयुक्त हैं, इस भाषा के सोवर समुदाय में लगातार बढ़ते उपयोग के मुख्य कारण हैं।

हमारे समुदाय में [शामिल हों](#) और अपने जावा साक्षात्कार के अनुभव साझा करें।

अनुशंसित ट्यूटोरियल:

[जावा ट्यूटोरियल](#)

[पहेलि](#)

[कोडिंग साक्षात्कार प्रश्न](#)

[जावा 8 साक्षात्कार प्रश्न](#)

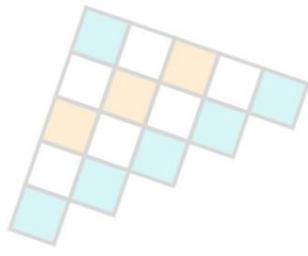
[जावा डेवलपर कैसे बनें?](#)

जावा फ्रेमवर्क:

[व्याज](#)

[हाइब्रिड](#)

[जावा एसई डाउनलोड](#)



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

एडब्ल्यूएस साक्षात्कार प्रश्न

गिट साक्षात्कार प्रश्न

जावा 8 साक्षात्कार प्रश्न मॉगोडब साक्षात्कार
प्रश्न

डीबीएमएस साक्षात्कार सवाल

स्प्रिंग बूट साक्षात्कार
प्रश्न

पावर बीआई साक्षात्कार प्रश्न

PI Sq| साक्षात्कार प्रश्न

ज्ञांकी साक्षात्कार
प्रश्न

लिनक्स साक्षात्कार प्रश्न

उत्तरदायी साक्षात्कार प्रश्न जावा साक्षात्कार प्रश्न

जेनकिंस साक्षात्कार प्रश्न