Practice OCA :

1. Mảng là các đối tượng vì vậy tham chiếu được thay đổi trong method gọi đến mảng đó được phản ảnh mọi nơi mà mảng được sử dụng.
2. Biến nguyên thủy được truyền giá trị , trong phương thức khi gọi tham số đã truyền một biến nguyên thủy khác với biến trong phương thức vì vậy thành viên thể hiện không thay đổi giá trị
3. Java.lang.Exception là một checked Exception có nghĩa là , phương thức throw ngoại lệ này phải khai báo nó trong mệnh đề throws do đó phương thức của bạn phải khai báo throws exception.
4. Object[] oa = new Object[10] , oa chứa 10 giá trị mặc định của object khai báo
5. Overload phương thức cùng tên khác tham số hoặc thứ tự , số tham số , kiểu trả về không quan trọng
6. RuntimeException ko cần phải handle trong khi Exception phải handle trong try catch or throws
7. Class chỉ khai bảo public khi trùng tên file.
8. Final class ko thể kế thừa , nhưng đc kế thừa class khác , phương thức final ko thể override nhưng có thể kế thừa , biến final ko thể thay đổi giá trị , nếu biến final trống đc khởi tạo bằng constructor và static final trống khởi tạo bằng static block
9. Static block code chạy trước rồi đến instance block rồi đến constructor , trong trg hợp kế thừa thì super luôn chạy trc nhưng ưu tiên nhất là static chạy trước rồi đến instance block code và constructor của super rồi đến subclass.
10. Phương thức static có thể dùng chung quản lý bởi class ko phải instance , biến static the same.
11. Từ khóa native chỉ dùng cho phương thức
12. Quy tắc Switch : Chỉ String , byte , char , short , int và các wrapper Class của nó , có cả enums được sử dụng làm kiểu một biến chuyển đổi , Chuỗi đc cho phép từ Java 7 . Các hằng số case phải được gán biến switch, nếu biến switch là String , biến case là String . Biến Switch phải đủ lớn để chứa tất cả các hằng số case , ví dụ biến char thì không có hằng số case nào vượt quá 65535 vì scope của char là 0 – 65535. Tất cả các biến case phải là constants. Không có hai trong số các biểu thức hằng số case liên quan đến biến switch có thể cùng giá trị. Nhiều nhất một biến default có thể liên kết với biến switch.
13. Mảng là một đối tượng , nhưng mảng của đối tượng không thể là mảng của các đối tượng khác. Object[] obj = new int[10] là sai. Size của mảng phải được khai báo bên phải. Int [] i ,j là I , j đều là mảng int , int I [] , j là I là mảng int còn j là số int.
14. Mảng đa chiều , size đầu tiên phải được chỉ định , các size sau có thể không cần chỉ định vì mảng đa chiều ko cần phải đối xứng. Mảng một chiều độ dài phải được chỉ định.
15. Các biến chứa tham chiếu đến các đối tượng thực tế . Việc chỉ định cho một tham chiếu đến đối tượng thực tế khác chỉ thay đổi giá trị của tham chiếu và không thay đổi giá trị đối tượng thực tế , toString override tại lớp String chỉ in ra chuỗi thực tế.
16. Bắt đầu từ java 7 bạn có thể sử dụng dấu gạch dưới giữa các số để giúp viết số dài dễ hơn ví dụ nếu viết 1000000 có thể viết 1\_000\_000. Lưu ý bạn không thể bắt đầu hoặc kết thúc một giá trị với dấu gạch dưới do đó 100\_ và \_100 là các giá trị không hợp lệ, \_100 là tên biến hợp lệ , bạn có thể sử dụng dấu gạch dưới cho tất cả các loại số bao gồm long , double , binary , float , hex ví dụ : int hex = 0xCAFE\_BASE , float f = 9898\_7878.333\_333f , int bin = 0b111\_0000\_1100\_1100;
17. Biến nào (hoặc phương thức tĩnh) sẽ được sử dụng tùy thuộc vào lớp mà biến được khai báo. Phương thức cá thể nào sẽ được sử dụng phụ thuộc vào lớp thực tế của đối tượng được tham chiếu bởi biến.
18. String trong cùng một lớp cùng 1 package hoặc trong các lớp khác nhau cùng 1 package hoặc các lớp khác nhau trong các package khác nhau biểu thị các tham chiếu đến cùng một đối tượng String , Các chuỗi được tính bằng các biểu thức không đổi được tính toán tại thời điểm biên dịch và sau đó được xử lý nếu chúng là chữ. Chuỗi được tính toán trong thời gian chạy run time , được tạo mới và do đó nó là khác biệt, Kết quả của việc thực hiện intern() một chuỗi là cùng một chuỗi bất kỳ chuỗi kí tự có sẵn nào có cùng nội dung.
19. Thực hiện toán tử luôn thực hiện bên trái trước rồi đến bên phải.
20. String , StringBuilder , StringBuffer đều là các final class , Boolean , Integer , Long , Short là final , Number không phải là final , int , long , double extended Number , System là final
21. Có các trường hoặc phương thức mơ hồ không tự gây ra bất kỳ vấn đề nào nhưng việc tham khảo các trường / phương thức đó theo cách không rõ ràng sẽ gây ra lỗi thời gian biên dịch. Vì vậy, bạn không thể gọi: System.out.println (VALUE); bởi vì nó sẽ mơ hồ (có hai định nghĩa GIÁ TRỊ). Nhưng các dòng sau là hợp lệ: TestClass tc = new TestClass (); System.out.println (((T1) tc) .VALUE); Tuy nhiên, không cần phải sử dụng cast rõ ràng để gọi phương thức m1 (): ((T2) tc) .m1 (); tc.m1 () cũng tốt vì mặc dù m1 () được khai báo trong cả interface, định nghĩa cho cả hai đều giải quyết rõ ràng chỉ một m1 (), được định nghĩa trong TestClass.
22. Để ghi đè một phương thức trong lớp con, phương thức ghi đè (nghĩa là phương thức trong lớp con) PHẢI CÓ: cùng một kiểu trả về trong trường hợp nguyên thủy (một lớp con được phép cho các lớp, đây còn được gọi là kiểu trả về covariant đối với trường hợp là đối tượng). cùng loại và thứ tự của các tham số. nó chỉ có thể ném ra những ngoại lệ được khai báo trong mệnh đề ném của phương thức của siêu lớp hoặc các ngoại lệ là các lớp con của các ngoại lệ được khai báo. Nó cũng có thể chọn KHÔNG ném bất kỳ ngoại lệ. Tên của các loại tham số không quan trọng. Ví dụ, void methodX (int i) giống như void methodX (int k)
23. Một lớp có thể có một phương thức tên là Main. Mặc dù, vì nó không giống như main, nên nó sẽ không được coi là phương thức chính tiêu chuẩn mà JVM có thể gọi khi chương trình được thực thi.
24. bạn nên nhớ rằng loại kết quả sẽ là int ngay cả khi cả hai toán hạng đều thuộc loại nhỏ hơn một số nguyên Do đó, sau đây sẽ không biên dịch - byte b1 = 1; byte b2 = 2; byte b = b1 + b2; // kết quả là kiểu int, không thể gán trực tiếp cho một byte Bạn phải sử dụng cast: byte b = (byte) (b1 + b2); // OK ngay Tương tự, khi một trong các toán hạng có kiểu long, float hoặc double và toán hạng khác có kích thước nhỏ hơn, kết quả sẽ lần lượt là long, float hoặc double.
25. RuntimeException là các ngoại lệ xảy ra bên trong chương trình , CheckedException là các ngoại lệ xảy ra bên ngoài chương trình, trong trường hợp một chương trình gọi đến một phương thức có đoạn code viết sai cố gắng trỏ đến một giá trị null , khi đó sử dụng RuntimeException là hợp lệ vì nó xảy ra bên trong chương trình và không lường trước đc nó cần được xác định trong khi Test và không nên xảy khi chương trình sẵn sàng để triển khai , trong trường hợp một chương trình gọi đến một phương thức truy cập tới một tệp và nó có thể không triển khai khi tệp không tồn tại do đó nó lường trước đc tình huống này và lỗi do một nguồn bên ngoài nên sử dụng CheckedException là hợp lệ.
26. Trình khởi tạo static hoặc cá thể block code chỉ ném ra RuntimeException, nếu bạn cố gắng ném một CheckedException từ một block code khởi tạo static hoặc cá thể ra bên ngoài, mã sẽ không compile. Bất kỳ ngoại lệ nào được ném trong một block static được gói vào ExceptionInitializerError và ném ra ExceptionInitializerError.
27. Đối với package không có tên thì không thể truy cập vào Class nằm trong package đó vì không có cách nào để import , đồng thời các thành phần Static cũng không thể import được.
28. Các biến Static có thể được để lại mà không cần khởi tạo rõ ràng và nó sẽ nhận được các giá trị mặc định , các biến final phải được khởi tạo rõ ràng riêng đối với static final phải được khởi tạo rõ ràng và có thể đặt trong static block code không thể đặt trong hàm khởi tạo vì static có thể đc gọi trước khi có thể hiển của một lớp được tạo nên nó không thể đợi cho đến khi hàm khởi tạo được gọi.
29. Garbage Collection: một đối tượng đủ điều kiện để Garbage Collection khi đối tượng đó không có bất kì một tham chiếu nào , bạn không thể đề xuất JVM gọi trực tiếp Garbage Collection nhưng bạn có thể đề xuất Garbage Collection bằng lệnh System.gc().
30. Đối với giá trị được protected , một lớp khác package extend Class chứa biến protected sẽ chỉ kế thừa biến này mà không thể đọc hoặc sửa đổi giá trị của biến của một thể hiện Class.
31. Ví dụ : boolean a; int b = 10 ; system.out.print(a = b ==10); sẽ in ra true vì giá trị b == 10 được thực hiện trc trả về true rồi gắn vào biến a; kiểu của biến sẽ được khai báo trước rồi được gán giá trị sau.
32. Các biến local phải được khởi tại bằng cách gán giá trị cho nó nếu có sử dụng đến nó nếu không trình biên dịch sẽ báo lỗi.
33. Chỉ cho phép thu hẹp ngầm đối với các biến nguyên thủy byte , char,short , int đối với biến long , double, float thì phải ép kiểu. Giá trị mặc định của long, double, float là một số int.
34. Interfaces , classes và enums đều là “Types” (Kiểu). Java cho phép một lớp triển khai nhiều interfaces. Bằng cách này , Java hỗ trợ đa kế thừa. Mặt khác, “State” (Trạng thái) được biểu diễn bằng các trường instance. Chỉ một lớp mới có thể có các trường thể hiện và do đó , chỉ một lớp mới có thể có trạng thái. (Các trường được xác định trong một interface luôn ở trạng thái static , ngay cả khi bạn không chỉ định từ khóa static một cách rõ ràng. Do đó , một interface không có bất kỳ trạng thái nào.) Vì một lớp chỉ được phép extend từ một lớp , nó chỉ có thể kế thừa một trạng thái. Do đó , Java không hỗ trợ đa kế thừa trạng thái. Trong java 8 cho phép một interface chứa các triển khai ở dạng phương thức mặc định và phương thức static. Tuy nhiên , Java 8 vẫn không cho phép một lớp kế thừa nhiều triển khai của cùng một phương thức từ hai giao diện không liên quan khác nhau. Ví dụ:

interface Movable{  
   public default void move(int dx) { System.out.println("in m");}  
}  
interface Growable{  
   public default void move(int dx) { System.out.println("in g");}  
}  
  
class Animal implements Movable, Growable{  
}

(Animal thực hiện Movable , Growable trình biên dịch sẽ phàn nàn rằng class Animal kế thừa các giá trị mặc định không liên quan cho move(int) từ các kiểu Movable và Growable. Để biên dịch nó bạn phải cung cấp một cách thực thi phương thức move(int) một cách rõ ràng trong lớp Animal.)

1. Khi khởi tạo một thể hiện tới một đối tượng lớp con kế thừa một lớp cha thì phải đảm bảo lớp cha được khởi tạo trước nên constructor của lớp con sẽ mặc định gọi tới super() của lớp cha ở dòng đầu tiên , trong trường hợp lớp cha ko có hàm constructor no-arg thì super() sẽ fail và cần phải gọi constructor của lớp cha ngay trong dòng đầu tiên của constructor của lớp con , có thể gọi tới constructor của lớp con ngay dòng đầu tiên nhưng phải đảm bảo sau đó constructor của lớp cha phải được xảy ra trước.
2. Tính đa hình làm cho code năng động hơn vì nó quyết định phương thức nào sẽ được gọi trong run time phụ thuộc vào lớp thức tế của đối tượng mặt khác nó làm cho code tái sử dụng hơn.
3. Ko thể ghi đè phương thức private và phương thức static của lớp cha , lớp con thỏa mái có phương thức static cùng tên với phương thức private và static của lớp cha.Ngoài ra phương thức thành phần có thể cùng tên với phương thức private của lớp cha.
4. Lưu ý rằng câu lệnh switch so sánh đối tượng String trong biểu thức của nó với các biểu thức được liên kết với mỗi nhãn trường hợp như thể nó đang sử dụng phương thức String.equals; do đó, việc so sánh các đối tượng String trong các câu lệnh switch là phân biệt chữ hoa chữ thường. Trình biên dịch Java thường tạo ra bytecode hiệu quả hơn từ các câu lệnh switch sử dụng các đối tượng String hơn là từ các câu lệnh if-then-else được xâu chuỗi.
5. Đây là quy tắc được đưa ra trong JLS liên quan đến việc gán các giá trị không đổi cho các biến nguyên thủy mà không cần ép kiểu rõ ràng:

Một chuyển đổi nguyên thủy thu hẹp có thể được sử dụng nếu tất cả các điều kiện sau được thỏa mãn:

Biểu thức là một biểu thức hằng thời gian biên dịch kiểu byte, char, short hoặc int.

Kiểu của biến là byte, short hoặc char.

Giá trị của biểu thức (được biết tại thời điểm biên dịch, vì nó là một biểu thức hằng) có thể biểu diễn trong kiểu của biến.

Lưu ý rằng chuyển đổi thu hẹp hoàn toàn (tức là chuyển đổi không có diễn viên rõ ràng) không áp dụng cho float, long hoặc double.

Ví dụ, char ch = 30L; sẽ không biên dịch được mặc dù 30 là đủ nhỏ để vừa với một ký tự.

1. Cố gắng ghi đè một phương thức tĩnh bằng một phương thức không tĩnh (và ngược lại) trong một lớp sẽ dẫn đến lỗi biên dịch. Ngay cả trong trường hợp giao diện, một giao diện con không thể ghi đè một phương thức mặc định bằng một phương thức tĩnh. Tuy nhiên, bạn có thể có một phương thức mặc định trong một giao diện con có cùng chữ ký với một phương thức tĩnh của siêu giao diện bởi vì một phương thức tĩnh của một giao diện chỉ có thể được gọi bằng tên của giao diện đó.
2. Hãy nhớ rằng mảng args không bao giờ rỗng. Nếu chương trình được chạy mà không có bất kỳ đối số nào, args trỏ đến một mảng Chuỗi có độ dài bằng 0. Do đó, hasParams sẽ là true và nó sẽ in ra "has params". Vì không có gì khác, khối mã tiếp theo cũng sẽ được thực thi và nó sẽ in ra "no params". Lưu ý rằng không sai về mặt cú pháp khi có phần mã được bao bọc trong {}.
3. Không thể sử dụng keyword cho label.
4. Tất cả các toán hạng kiểu byte, char hoặc short được thăng cấp ÍT NHẤT thành int trước khi thực hiện các phép toán. Nếu một trong các toán hạng lớn hơn số int thì toán hạng còn lại được thăng cấp thành cùng kiểu. Lưu ý rằng System.out.println ((float) 5/4); sẽ in 1,25. Nếu bạn loại bỏ ép kiểu rõ ràng (float), nó sẽ in ra 1.
5. Đóng gói là kỹ thuật được sử dụng để đóng gói thông tin theo cách để che giấu những gì cần được ẩn và làm cho những gì được dự định sẽ hiển thị. Nói một cách dễ hiểu, đóng gói thường có nghĩa là làm cho các biến dữ liệu ở chế độ riêng tư và cung cấp các trình truy cập công khai.
6. Đây là những gì JLS nói về điều này:

1 Đánh giá toán hạng tay trái đầu tiên

2 Đánh giá toán hạng trước khi hoạt động

3 Đánh giá Tôn trọng Dấu ngoặc đơn và Thứ tự ưu tiên

4 Danh sách đối số được đánh giá từ trái sang phải

Đối với Mảng: Đầu tiên, các biểu thức thứ nguyên được đánh giá, từ trái sang phải. Nếu bất kỳ đánh giá biểu thức nào hoàn thành đột ngột, các biểu thức ở bên phải nó sẽ không được đánh giá.

46.Cơ chế Exception của java giúp chúng ta có thể xác định các ngoại lệ của riêng mình dựa trên miền doanh nghiệp ứng dụng của bạn. Ví dụ: trong một ứng dụng ngân hàng, bạn có thể muốn tạo một InsuffnoughFundsException. Điều này làm tăng sự rõ ràng của mã so với việc có một (hoặc một vài) lớp ngoại lệ duy nhất (hoặc một vài tiêu chuẩn) và xem mã ngoại lệ để xác định điều gì đã xảy ra. ngoài ra , Logic xử lý lỗi được đưa vào khối catch, làm cho luồng chính của chương trình trở nên sạch sẽ và dễ hiểu.

47. Vòng lặp for nâng cao cần một mảng hoặc một đối tượng của lớp triển khai java.lang.Iterable. Bản đồ không triển khai Iterable, mặc dù bạn có thể sử dụng các phương thức keySet () hoặc các giá trị () để lấy một Bộ sưu tập (mở rộng lặp lại) và sau đó lặp lại Bộ sưu tập đó.

1. 'this' được JVM gán một tham chiếu đến đối tượng hiện tại tự động. Do đó, trong một phương thức thể hiện foo, gọi this.foo (); giống như cách gọi foo (); Vì không có đối tượng hiện tại nào có sẵn cho một phương thức tĩnh, nên tham chiếu 'this' không có sẵn trong các phương thức tĩnh và do đó nó chỉ có thể được sử dụng trong các phương thức cá thể. Vì lý do tương tự, các phương thức tĩnh không thể truy cập trực tiếp các trường hoặc phương thức không phải tĩnh của lớp đó, tức là không có tham chiếu đến một thể hiện của lớp đó. Lưu ý: bạn không thể gán lại 'this' như thế này: this = new Object ();
2. Khái niệm cần hiểu ở đây như sau:

Nếu trình biên dịch có thể tìm ra điều gì đó KHÔNG BAO GIỜ có thể xảy ra, thì nó sẽ gắn cờ báo lỗi. Trong câu hỏi này, trình biên dịch biết rằng ln, i hoặc d không bao giờ có thể trỏ đến cùng một đối tượng trong bất kỳ trường hợp nào vì chúng là các tham chiếu đến các lớp khác nhau của các đối tượng không có mối quan hệ (lớp cha / lớp con) giữa chúng

50.Lệnh gọi printSum (1, 2) sẽ được liên kết với printSum (int, int) vì 1 và 2 là int, khớp chính xác với int, int. Lưu ý rằng nếu phương thức printSum (int, int) không có trong mã, printSum (double, double) sẽ được gọi thay vì printSum (Integer, Integer) vì việc mở rộng được ưu tiên hơn so với boxing / unboxing. Chúng tôi khuyên bạn nên chạy chương trình này và thử các kết hợp khác nhau. Đề thi có các câu hỏi về mẫu này.

51.Một lớp được tải khi nó được sử dụng lần đầu tiên.

52.Hàm tạo mặc định được định nghĩa trong một class khi trong class đó không có xác định bất kỳ hàm tạo nào kể cả hàm tạo giống với hàm tạo mặc định. Ngoài ra hàm tạo mặc định không có tham số chính thức và không có mệnh đề throw nào được khai báo ngầm , nếu lớp được khai báo là Object của lớp nguyên thủy thì phương thức khởi tạo mặc định có phần thân trống. Hàm tạo mặc định đơn giản gọi hàm tạo lớp cha mà không có đối số, nó sẽ lỗi complie nếu hàm tạo mặc định được khai báo ngầm nhưng lớp cha không có hàm tạo có thể truy cập.Nếu hàm tạo của lớp cha có mệnh đề throw thì lỗi complie sẽ xảy ra.

53. Bạn cần nhớ những điểm sau về Boolean: 1. Lớp Boolean có hai hàm tạo - Boolean (String) và Boolean (boolean) Hàm tạo String cấp phát một đối tượng Boolean đại diện cho giá trị true nếu đối số chuỗi không null và bằng nhau, bỏ qua trường hợp, đối với chuỗi "true". Nếu không, hãy cấp phát một đối tượng Boolean đại diện cho giá trị false. Ví dụ: new Boolean ("True") tạo ra một đối tượng Boolean đại diện cho true. new Boolean ("yes") tạo ra một đối tượng Boolean đại diện cho false. Hàm tạo boolean tự giải thích. 2. Lớp boolean có hai phương thức trợ giúp tĩnh để tạo boolean - parseBoolean và valueOf. Phương thức Boolean.parseBoolean (String) trả về một boolean nguyên thủy chứ không phải đối tượng Boolean (Lưu ý - Tương tự với trường hợp của các phương thức parseXXX khác như Integer.parseInt - chúng trả về nguyên thủy chứ không phải đối tượng). Boolean được trả về đại diện cho giá trị true nếu đối số chuỗi không null và bằng nhau, bỏ qua chữ hoa, đối với chuỗi "true". Mặt khác, Boolean.valueOf (String) và phiên bản Boolean.valueOf (boolean) overload của nó hoạt động tương tự nhưng trả về một tham chiếu đến các đối tượng wrapper Boolean.TRUE hoặc Boolean.FALSE. Quan sát rằng họ không tạo một đối tượng Boolean mới mà chỉ trả về các hằng số tĩnh TRUE hoặc FALSE được xác định trong lớp Boolean. 3. Khi bạn sử dụng toán tử bình đẳng (==) với boolean, nếu chính xác một trong các toán hạng là wrapper Boolean, thì đầu tiên nó được unboxed thành một nguyên thủy boolean và sau đó hai toán hạng được so sánh (JLS 15.21.2). Nếu cả hai đều là wrapper Boolean, thì các tham chiếu của chúng được so sánh giống như trong trường hợp của các đối tượng khác. Do đó, new Boolean ("true") == new Boolean ("true") là false, nhưng new Boolean ("true") == Boolean.parseBoolean ("true") là true.

54.Lưu ý rằng có một vài câu hỏi trong bài kiểm tra kiến thức của bạn về cách các thông báo ngoại lệ được in. Khi bạn sử dụng System.out.println (ngoại lệ), dấu vết ngăn xếp không được in. Chỉ cần tên của lớp ngoại lệ và thông báo được in. Khi bạn sử dụng exception.printStackTrace (), một chuỗi hoàn chỉnh gồm tên của các phương thức được gọi, cùng với số dòng, được in ra. Nó chứa tên của các phương thức trong chuỗi các lệnh gọi phương thức dẫn đến nơi tạo ngoại lệ quay ngược lại điểm bắt đầu luồng, trong đó ngoại lệ được tạo, bắt đầu.

55. Chỉ + là quá tải đối với Chuỗi. a + = x thực sự được chuyển đổi thành a = a + x. vì vậy nó hợp lệ cho Chuỗi. toán tử dot (.) truy cập các thành viên của đối tượng String. Tuy nhiên, chỉ có một biến thành viên: CASE\_INSENSITIVE\_ORDER. Nó thuộc loại Comparator (là một giao diện).

56. Nếu biểu thức tham chiếu mảng tạo ra null thay vì tham chiếu đến một mảng, thì một NullPointerException được ném vào thời gian chạy, nhưng chỉ sau khi tất cả các phần của biểu thức tham chiếu mảng đã được đánh giá và chỉ khi các đánh giá này hoàn thành bình thường. Điều này có nghĩa là, index đầu tiên = 2 sẽ được thực thi, index này sẽ gán 2 cho chỉ mục. Sau khi null [2] đó được thực thi, nó sẽ ném ra một NullPointerException. Nhưng ngoại lệ này bị chặn bởi khối catch, khối này không in ra. Vì vậy, có vẻ như NullPointerException không được ném nhưng thực tế là như vậy.

Nói cách khác, việc gán 2 cho chỉ mục được nhúng xảy ra trước khi kiểm tra tham chiếu mảng do getArray () tạo ra.

Trong một truy cập mảng, biểu thức ở bên trái của dấu ngoặc dường như được đánh giá đầy đủ trước khi bất kỳ phần nào của biểu thức trong dấu ngoặc được đánh giá. Lưu ý rằng nếu việc đánh giá biểu thức bên trái dấu ngoặc hoàn thành đột ngột, không có phần nào của biểu thức trong dấu ngoặc dường như đã được đánh giá.

57. class A{

A() { print(); }

void print() { System.out.println("A"); }

}

class B extends A{

int i = 4;

public static void main(String[] args){

A a = new B();

a.print();

}

void print() { System.out.println(i); }

}

Lưu ý rằng phương thức print () được ghi đè trong lớp B. Do tính đa hình, phương thức được thực thi được chọn tùy thuộc vào lớp của đối tượng thực. Ở đây, khi một đối tượng của lớp B được tạo, phương thức khởi tạo mặc định đầu tiên của B (không hiển thị trong mã nhưng được trình biên dịch tự động cung cấp vì B không xác định rõ ràng bất kỳ phương thức khởi tạo nào) được gọi. Dòng đầu tiên của hàm tạo này là một lời gọi tới super (), gọi hàm tạo của A. Phương thức khởi tạo của A lần lượt gọi print (). Bây giờ, print là một phương thức thể hiện không riêng tư và do đó là đa hình, có nghĩa là, việc lựa chọn phương thức sẽ được thực thi phụ thuộc vào lớp của đối tượng thực tế mà nó được gọi. Ở đây, vì lớp của đối tượng thực tế là B nên bản in của B được chọn thay vì bản in của A. Tại thời điểm này, biến i chưa được khởi tạo (vì chúng ta vẫn đang trong quá trình khởi tạo A), vì vậy giá trị mặc định của nó, tức là 0 được in. Cuối cùng, 4 được in.

58. Lưu ý rằng không có công cụ sửa đổi nào cho hàm tạo của A. Vì vậy, nó có quyền truy cập mặc định. Điều này có nghĩa là chỉ các lớp trong gói a mới có thể sử dụng nó. Cũng lưu ý rằng lớp B nằm trong một gói khác và được mở rộng từ A. Trong hàm khởi tạo của B, trình biên dịch sẽ tự động thêm super () làm dòng đầu tiên. Nhưng vì A () không thể truy cập được trong B, mã này sẽ không biên dịch.

59. super.methodName là một cách hợp lệ để gọi phương thức của lớp siêu từ bất kỳ đâu trong phương thức của lớp con. Nhưng nó chỉ hoạt động cho các lớp học. Để gọi phương thức mặc định của giao diện, bạn cũng cần sử dụng tên của giao diện đó. Như thế này: Account.super.getId (); Một lớp (hoặc một giao diện) có thể gọi một phương thức mặc định của một giao diện được đề cập rõ ràng trong mệnh đề triển khai của lớp (hoặc mệnh đề mở rộng của giao diện) bằng cách sử dụng cú pháp tương tự, tức là <Tên giao diện> .super. <methodName>. Tuy nhiên, kỹ thuật này không thể được sử dụng để gọi một phương thức mặc định được cung cấp bởi một giao diện không được thực thi trực tiếp (hoặc mở rộng) bởi người gọi. Ví dụ:

interface A {  
   default void hello() {  
   }  
}  
  
interface B extends A {  
   default void hello() {  
       super.hello();    //This is NOT valid.  
       A.super.hello();    //This is valid.  
   }  
}  
  
public class TestClass implements B {  
   public void hello() {  
      super.hello();//This is NOT valid.  
      A.super.hello(); //This is NOT valid because TestClass does not implement A directly.  
      B.super.hello(); //This is valid.  
   }  
}

60. Lưu ý rằng các toán tử boolean có nhiều ưu tiên hơn =. (Trên thực tế, = có ít ưu tiên nhất trong tất cả các toán tử.)

vì vậy, trong (b2! = b1 =! b2) b2! = b1 đầu tiên được đánh giá trả về giá trị 'false'. Vì vậy biểu thức trở thành false =! B2. Và điều này là bất hợp pháp vì false là một giá trị chứ không phải một biến!

Nếu nó là một cái gì đó giống như (b2 = b1! = B2) thì nó là hợp lệ vì nó sẽ thành: b2 = false.

61.

String str1 = "one"; String str2 = "two";

System.out.println( str1.equals(str1=str2) );

Đầu tiên giá trị của 'str1' được đánh giá (tức là “one”). Bây giờ, trước khi phương thức được gọi, các toán hạng được đánh giá, vì vậy str1 trở thành "two". vì vậy "one" .equals ("two") là sai.

62. Functional programming interface Predicate có hàm test(T t) nó nhận một tham số duy nhất và trả về một boolean. Lambda expres-sions có một tham số được phép bỏ qua dấu ngoặc đơn xung quanh danh sách các tham số.Câu lệnh nằm trong body , lệnh return phải được sử dụng nếu dấu ngoặc nhọn được bao gồm xung quanh phần body.Autoboxing hoạt động đối với các Collection không dành cho Predicate.

63.Một lớp hoặc kiểu interface T sẽ được khởi tạo ngay lập tức trước lần xuất hiện đầu tiên của bất kì điều sau đây:

- T là một lớp và một thể hiện của T được tạo ra.

- T là một lớp và một phương thức static do T khai báo được gọi.

- Một trường static do T khai báo được gán.

- Một trường static do T khai báo được sử dụng và trường không phải là một biến hằng số.

- T là một lớp cấp cao nhất và một câu lệnh khẳng định được lồng từ vựng trong T được thực thi.

Tham chiếu đến trường static chỉ gây ra việc khởi tạo lớp hoặc interface thực sự khai báo nó , mặc dù nó có thể được tham chiếu qua tên của lớp con , giao diện con hoặc lớp triển khai giao diện.

Việc gọi các phương thức phản chiếu nhất định trong lớp Class và trong package java.lang.reflect cũng gây ra việc khởi tạo lớp hoặc interface.

Một lớp hoặc interface sẽ không được khởi tạo trong bất kỳ trường hợp nào khác.

64.Lưu ý , không được phép cho toán hạng thứ hai và thứ ba của toán tử ? : là một lệnh gọi của một phương thức void.

Kiểu của biểu thức được xây dựng bằng cách sử dụng ? : được xác định bởi kiểu của toán hạng thứ hai và thứ ba.

Nếu một trong các toán hạng có kiểu byte và toán hạng còn lại thuộc kiểu short thì kiểu của biểu thức điều kiện là short.

Nếu một trong các toán hạng thuộc kiểu T trong đó T là byte , short hoặc char, và toán hạng còn lại là một biểu thức hằng số kiểu int có giá trị có thể biểu diễn trong kiểu T , thì kiểu của biểu thức điều kiện là T.

Mặt khác , thăng hạng số nhị phân được áp dụng cho các kiểu toán hạng và kiểu của biểu thức điều kiện là kiểu thăng hạng của toán hạng thứ hai và thứ ba.

Nếu một trong các toán hạng thứ hai và thứ ba thuộc kiểu null và kiểu của toán hạng kia là kiểu tham chiếu, thì kiểu của biểu thức điều kiện là kiểu tham chiếu đó.

Nếu toán hạng thứ hai và thứ ba thuộc các kiểu tham chiếu khác nhau, thì phải có thể chuyển đổi một trong các kiểu này sang kiểu khác(gọi kiểu sau này là T) bằng cách chuyển đổi phép gán; kiểu của biểu thức điều kiện là T. Đây là lỗi compile nếu không có kiểu gán nào tương thích với kiểu kia.

Lưu ý rằng: số nhị phân thực hiện chuyển đổi unboxing và chuyển đổi tập giá trị.

65.

public class DatabaseWrapper {

static String url = "jdbc://derby://localhost:1527//mydb";

static DatabaseWrapper getDatabase() {

System.out.println("Getting DB");

return null;

}

public static void main(String[ ] args) {

System.out.println( getDatabase().url );

}

}

Tham chiếu null có thể được sử dụng để truy cập một biến lớp static mà không gây ra ngoại lệ. Lưu ý signature của phương thức.Nó trả về một tham chiếu đến một đối tượng của lớp DatabaseWrapper. Do đó , getDatabase().Url có nghĩa là chúng ta đang truy cập vào trường Url của đối tượng được phương thức trả về.Bây giờ , vì lớp của đối tượng được phương thức trả về là DatabaseWrapper và trường Url là trường static của lớp , trình biên dịch tạo ra lệnh cho JVM truy cập trực tiếp trường này bằng cách sử dụng tham chiếu lớp thay vì tham chiếu đối tượng được trả về phương thức trong thời gian chạy.Do đó , JVM không cần phụ thuộc vào đối tượng thực tế được phương thức trả về tại thời điểm chạy để truy cập Url.Vì vậy , ngay cả khi phương thức trả về null tại thời điểm chạy, điều đó không quan trọng vì JVM thậm chí không truy cập tham chiếu được phương thức trả về.

66.

Hãy quan sát rằng làm tròn là một quy trình toán học tiêu chuẩn trong đó số nằm chính xác giữa hai số luôn làm tròn đến số cao hơn. Vì vậy, 0,5 vòng thành 1 và -,5 vòng thành 0.

67.

Các toán tử gán ghép (+= , \*= , …) có những cách khác , biểu thức gán ghép có dạng E1 op = E2 tương đương với E1 = (T) ((E1) op (E2)) , trong đó T là loại E1 , ngoại trừ E1 chỉ được đánh giá một lần. Lưu ý rằng , phép ép kiểu ngụ ý sang kiểu T có thể là một chuyển đổi danh tính hoặc một chuyển đổi nguyên thủy thu hẹp. Ví dụ: short x = 3 ; x+ = 4.6 và kết quả là x có giá trị 7 vì nó tương đương với : short x = 3 ; x = (short) (x + 4.6);

68.

Tất cả các đối tượng của Wrapper đều là immutable object.Vì vậy, khi bạn thực hiện obj++ những gì thực sự xảy ra là như sau: obj = new Integer(obj.intValue()+1)