1. Cevap C. protected erişim belirleyicisi alt sınıfların, üst sınıfların protected elementlerine erişimine izin verir. protected olarak belirlenen metot ve özelliklere alt sınıflardan veya aynı paketteki sınıflardan ulaşılır. package-private(default) erişim belirleyicisi aynı pakette yer alan sınıflara erişim izni sağlar.
   Kaynak: http://mail.baskent.edu.tr/~tkaracay/etudio/ders/prg/java/ch11/accessmods.htm
   https://www.linkedin.com/pulse/encapsulationkaps%C3%BClleme-ve-java-eri%C5%9Fim-access-modifiers-koray-peker/?originalSubdomain=tr
2. Cevap B. Java'da üst sınıfın kurucu metoduna erişim sağlamak için super() metodu, mevcut sınıfın kurucu metoduna erişim sağlamak için this() metodu kullanılır. C ve D seçeneklerinde verilen ifadeler kurucu metot değillerdir.
   Kaynak: https://www.geeksforgeeks.org/difference-super-java/
   http://mail.baskent.edu.tr/~tkaracay/etudio/ders/prg/java/ch15/super.htm
3. Cevap D. sell() metodunda kullanılan if-else if kontrol yapısında bir else bloğu yer almamaktadır. Bu metodun return ifadelerinin koşul bloklarında yer alması nedeniyle kod derlenmez. Çünkü, metodun işletimi sırasında if veya else if blok gövdeleri çalışmazsa metot return yapamaz. Sell() metodunun son satırında return true ya da return false komutu eklenirse price değişkeni 3 defa arttırılarak 8 değerini ekrana basacaktı.
4. Cevap D. Verilen kod örneği metot aşırı yükleme (method overloading) örneğidir. main() metodunda çağrılan nested() metodu parametresizdir ve Dolls sınıfındaki parametresiz metottur. Bu metodun dönüş tipi void olduğu için return yapmaz ve bu nedenle kod derlenmez. Eğer main() metodunda çağrılan nested() metoduna int tipinden bir değer gönderirsek int parametreli olan nested(int) metodunu çağıracaktır ve ekrana bu değerin 1 fazlasını yazacaktır. Eğer nested(int, boolean) metodunu çağırırsak bu metodun kendi içinde int parametreli metodu çağırması nedeniyle yine verilen değerin 1 fazlasını yazacaktır.
   Kaynak: https://www.javatpoint.com/method-overloading-in-java
   https://www.injavawetrust.com/pure-java-13-object-orientation-04-overloaded/
5. Cevap B. Java bir metoda nesnelerin primitiflerini veya referanslarını geçirirken pass-by-value kullanır.
   pass-by-value
   Metoda gönderilen parametrenin bir kopyasının tutularak gönderildiği parametre aktarım şeklidir. Yani metot içinde parametrenin değeri değiştiğinde ilk değeri değişmez. Değişiklik sadece metot scope içinde görülür.
   pass-by-reference
   Metoda gönderilen parametrenin referansının gönderildiği parametre aktarım şeklidir. Metot içinde parametrenin değeri değiştiğinde ilk değeri de değişir.
   Kaynak: https://www.javaworld.com/article/3512039/does-java-pass-by-reference-or-pass-by-value.html
   https://medium.com/bili%C5%9Fim-hareketi/javada-bellek-y%C3%B6netimi-c99d73657577
6. Cevap C.
   Basitçe Java Bean, tekrar kullanılabilir bir yazılım bileşenidir (reusable software component). Daha detaylı bakıldığında aslında her Java Bean bir ya da birden çok sınıftan (class) oluşmuş ve tek başına çalışma yeteneği olan bileşenlerdir.
   Tek başına çalışabilen bu bileşenler, daha gelişmiş programlar oluşturmakta kullanılırlar. Düşük bağlılık (coupling) ve yüksek uyumdaki (cohesion) program parçalarının bir araya getirilmesi ile daha modüler bir yaklaşım elde etmek ve büyük bir projeyi parçalara bölmek mümkündür.

Java Bean'lerin klasik nesne yönelimli modellemedeki (object oriented modelling) sınıf (class) bölmesinden farkı daha üst seviyeli olmaları ve nesnel bölmelerden daha çok kavramsal bölmelere gidilebilmesidir.

Oysaki bütün bu sınıfları birleştirerek tek bir Java Bean yapmak mümkündür. Tek başına çalışan bu Java Bean projenin bir modülü olup bu modül kullanılarak daha büyük sistemlerin inşası mümkündür.

Java Bean'lerin bir diğer özelliği ise geliştirme sürecinde çalıştırılabilir olmalarıdır. Sonuçta tek başına çalışan bu bileşenler, kod geliştirme (Development) zamanında da çalışabilir ve yazılımı geliştiren kişilere anlık olarak kullanma ve yaptığı her işlemi test etme imkânı sağlar. JAVA Beanler ayrıca şu 3 özelliği barındırmalıdır:

- Public Constructor
- Serializable olmalıdırlar yani Serializable arayüzünü (interface) uygulamalıdırlar (implements)
- Erişim metodları bulunmalıdır (getter /setter methods)

Yani bir Java Bean'in hiç constructor metoduna ihtiyacı olmasa bile bir adet boş yapıcı (empty constructor) bulunmalıdır. Ayrıca hiçbir değişkenin erişimi public olmamalıdır. Bunun yerine bu değişkene erişen getter ve setter fonksiyonları bulunmalıdır.

Verilen bilgiler ışığında Java Bean'lerin getter ve setter metotlarının imzaları aşağıdaki adreslerde yer almaktadır. getter bir metot bir erişim belirleyicisi, dönüş tipi ve get ile başlayan metot isminden oluşur. setter bir metot ise erişim belirleyicisi, void dönüş tipi ve set ile başlayan metot ismi ve set edilecek olan değişken tipinde bir parametreden oluşur. Bu bilgilere göre A ve B seçeneklerindeki metot tanımları geçerli birer Java Bean metot imzası değildir. D seçeneği tamamen Java Bean metot imzasından uzak bir metottur. Geçerli metot imzası C seçeneğindeki imzadır.

Kaynak: https://www.javatpoint.com/java-bean
https://docstore.mik.ua/orelly/java-ent/jnut/ch06_02.htm
http://bilgisayarkavramlari.sadievrenseker.com/2009/01/05/java-bean/

7. Cevap B. super() ve this() kurucu metot çağrıları, kurucu metodun ilk satırında olmak zorundadır. Bu nedenle A seçeneği doğrudur. super() ve this() kurucu metot çağrıları, aynı kurucu metot için kullanılamazlar. Bu nedenle B seçeneği yanlıştır. this() kurucu metot çağrısında argüman geçilmesi, argüman listesini barındıran bir kurucu metot gerektirir. Bu nedenle C seçeneği doğrudur. Argüman geçilmezse, parametresiz kurucu metoda çağrıda bulunacaktır. D seçeneği de doğrudur.
   Kaynak: https://www.geeksforgeeks.org/difference-super-java/
8. Cevap B. A seçeneğinde public erişim belirleyicisinin ilk harfi büyük olduğu için geçersiz bir tanımlamadır ve bu seçenek yanlıştır. compute() metodunun bir geri dönüşü olduğu için void olamaz. Bu nedenle C seçeneği de yanlıştır. D seçeneğinde ise String dönüş tipi gösterilmektedir. Ancak geri dönen değer nümerik olduğu için D seçeneği de yanlış olacaktır. B seçeneğindeki Long tipi bir referans tiptir. Aynı zamanda erişim belirleyici verilmediği için package-private (default) erişim belirleyicisine sahip olacaktır. Ancak bu durumda bile Long ifadesi boşluk tamamlandığında return ifadesi Long'a cast edilmelidir.
9. Cevap C. static olarak tanımlanan bir değişkenin bellekte tek bir kopyası bulunur ve sınıfın tüm örnekleri için kullanılabilir haldedir.
   Kaynak: https://www.javatpoint.com/static-keyword-in-java
   https://www.injavawetrust.com/pure-java-20-static-variables-and-methods/

10. Cevap A. p1 numaralı satıra this(4) kodu eklenirse parametreli kurucu metodu çağırarak static olan rope değişkenine, outside değişkeninin de true olması nedeniyle 4 değerinin 1 fazlası olan 5 değerini yükleyerek konsol ekranına çıktı olarak gönderecektir.
11. Cevap B.
    public: Fonksiyonunun dış kullanıcılara açık olduğu ve buna Java InterPreter'ı da dahil herkes tarafından erişebileceğimizi belirtilir.
    protected: Aynı paket içerisinden ve bu sınıftan türemiş alt sınıflar tarafından erişilmeyi sağlayan erişim belirleyicisi.
    friendly: Yalnızca aynı paket içerisinden erişilmeyi sağlayan erişim belirleyicisi.
    private: Yalnızca kendi sınıfı içerisinden erişilmeyi sağlayan, başka her yerden erişimi kesen erişim belirleyicisi.
    A, C ve D seçeneklerinde verilen bilgiler doğrudur. Ancak, bir sınıfın örneğinin üst sınıftaki package-private özellik ve metotlara erişebilmesi için aynı paket içinde yer alması gerektiği için B seçeneği yanlıştır.
    Kaynak: http://mail.baskent.edu.tr/~tkaracay/etudio/ders/prg/java/ch11/accessmods.htm
    https://www.dijitalders.com/icerik/44/2138/public_private_protected.html

    https://www.dijitalders.com/icerik/44/402/public_static_void_protected_friendly_private_nedir.html

12. Cevap D. Soruda verilen stuff değişkeni public olarak deklare edildiği için kapsülleme mantığına direkt aykırıdır. Kapsülleme ile Java Beans mantığı karıştırılmamalıdır. Eğer bir sınıfa ait değişkenler private yapılmışsa bu sınıf kapsüllenmiştir(encapsulated). Bu nedenle doğru cevap bunlardan hiçbirisidir.
    Kaynak: https://www.javatpoint.com/java-bean
    https://docstore.mik.ua/orelly/java-ent/jnut/ch06_02.htm
    http://bilgisayarkavramlari.sadievrenseker.com/2009/01/05/java-bean/
13. Cevap C. Java eğer hiç kurucu metot tanımlanmadıysa argümansız bir kurucu otomatik olarak tanımlar. Ancak, en az bir kurucu metot varsa herhangi bir kurucu metot tanımlamayacağı için A seçeneği yanlıştır. Üst sınıf derleyici tarafında oluşturulan ve alt sınıftan ulaşılabilen argümansız bir kurucuya otomatik sahip olduğundan dolayı B seçeneği de yanlıştır. Bir sınıf birden fazla argümansız kurucu metoda sahip olamaz. Çünkü, metot imzalarının aynı olması anlamına geleceği için D seçeneği de yanlıştır. Bir alt sınıf, argümansız kurucu metoda sahip bir üst sınıfı genişlettiğinde alt sınıf içinde argümansız kurucu metot otomatik olarak eklenmez. Bunun yerine geliştirici alt sınıf içinde bir kurucu metot tanımlayarak üst sınıfa çağrı yapılmasını sağlar.
    Kaynak: https://www.geeksforgeeks.org/constructors-in-java/
    https://docs.oracle.com/javase/tutorial/java/javaOO/constructors.html
14. Cevap A. Bir metot imzasında sadece bir varargs parametresi yer alabilir ve bu parametre argüman listesinde en sonda olmalıdır.
    Kaynak: https://www.geeksforgeeks.org/variable-arguments-varargs-in-java/
15. Cevap C. slalom() metoduna gönderilen Ski sınıfının örnek referansı, myName değişkeni ve mySpeed dizisi final olarak tanımlandığı için ilk yüklenen değerleri değiştirilemez. Bu nedenle slalom() metodunda racer isimli nesne referansı değişmese de bu örneğin instance değişkenleri değişir. Böylece age değeri 18 olarak güncellenirken, myName="Rosie" kalır, mySpeed[0]=0 olur. Bu kod örneğini demonstre eden görsel C seçeneğindekidir.
    Kaynak: https://www.geeksforgeeks.org/final-keyword-java/
16. Cevap B. Bir metodun aşırı yüklenmesi için aynı isimde ve farklı sayıda argüman içeren bir metot implemente edilmesi gerekir. A ve D seçeneklerindeki metotlar aynı parametrelere

sahip olması nedeniyle aşırı yüklenmiş (overloaded) metotlar değillerdir. C seçeneğindeki metodun ise ismi farklıdır ve bu kapsamda değerlendirilmeyeceği için C seçeneği de doğru değildir. B seçeneğindeki metodun dönüş tipi Long olsa dahi, Integer'ın Long'a yükselebilmesi dolayısıyla B seçeneği findAverage() metodunu aşırı yükler(overload).
Kaynak: https://www.geeksforgeeks.org/overloading-in-java/

17. Cevap D.
Nesne yönelimli programlamanın ilk prensibi kapsülleme (encapsulation) olarak adlandırılır. Bu özellik, dilin nesne kullanıcısından gereksiz uygulama ayrıntılarını saklar. Oluşturulan bir sınıf (class) içerisinde kullanıcının işlemlerini daha kolay gerçekleştirebilmesi için bazı işlemler birleştirilerek tek bir işlem gibi gösterilir. Bu birleştirme işlemine kapsülleme denir.
Erişim belirleyicileri (access modifier) sayesinde kapsülleme çok daha kolay yapılmaktadır. Erişim belirteçleri, oluşturulan sınıf veya sınıf içindeki elemanların erişim seviyelerini belirlemek için kullanılan anahtar kelimeler grubuna verilen isimdir. Metotlar ve değişkenler bir anahtar sözcük ile önceden belirlenen sınırlar dahilinde kullanılabilir. Bu anahtar kelimeler şu şekilde sıralanabilir.
public: Sistemdeki bütün sınıfların erişebilmesini sağlar. Yalnızca aynı proje içinden değil, diğer projelerden de erişim sağlanabilir.
private: Bir "özellik (property)"in veya "metod"un sadece tanımlandığı sınıftan erişilebilmesini sağlar. Oluşturulan sınıf veya yapıların "public" olması açık bir şekilde belirtilmez ise, derleyici tarafından "private" olarak belirlenir.
internal: Aynı derleyici (assembly) içinde bulunan tüm sınıflardan erişim sağlanır.
protected: Sadece tanımlandığı sınıfın içinde ve o sınıftan türetilmiş diğer sınıfların içinde erişilebilir.
Kapsülleme "private" değişkenlerin metotlar gibi kullanılmasına yardımcı olur. Okuma (Read Only) işleminin yanısıra okuma - yazma (read - write) işleminin yapılmasını sağlar.

A seçeneği doğrudur. Kapsülleme sayesinde diğer geliştiricilerin bu sınıfları kullanması daha kolaylaşır. B seçeneği doğrudur. Kapsülleme sayesinde özelliklerin private olarak deklare edilmesi ile veri bütünlüğü sağlar. C seçeneği doğrudur. Kapsüllenen özelliklerin doğrudan değiştirilmesi önlenmektedir. D seçeneği doğru değildir. Kapsülleme performans ve eşzamanlılığı garanti etmez. Zaten bu durum kapsüllemenin bir amacı değildir.
Kaynak: https://dzone.com/articles/why-encapsulation-matters
https://beginnersbook.com/2013/05/encapsulation-in-java/
https://bidb.itu.edu.tr/seyir-defteri/blog/2013/09/08/kaps%C3%BClleme-(encapsulation)

18. Cevap A. String değerler immutable'dır ve değiştirilemez. Bu nedenle B seçeneği yanlıştır. C ve D seçenekleri ilkel değişkenlerdir. Java'da referansa göre değil değere göre geçiş yapıldığı için C ve D seçenekleri de yanlıştır. Nesneye yapılan başvurunun bir kopyası geçirildiği için bir dizinin içeriği değiştirilebilir ve daha sonra bir yönteme geçirilebilir.
19. Cevap B. A seçeneğindeki ifade Java yazım kurallarına uymadığı için yanlıştır. C seçeneğindeki ifadenin geçerli olması için static import yapılması gerekmektedir. D seçeneği geçerli bir kullanımdır ancak en iyi yol değildir. Çünkü, aynı pakette yer alan sınıflar arasında metot çağrıları yaparken paket ismi verilmesi gerek yoktur. Bu nedenle en iyi kullanım şekli B seçeneğindeki gibidir.
Kaynak: https://www.geeksforgeeks.org/overloading-in-java/
20. Cevap D. Bir metodun bir dönüş tipi olduğu biliniyorsa A ve B seçeneklerindeki gibi bir veri tipi olabilir. return ifadesi kullanılmasını gerektirir. Eğer bir metodun dönüş tipi yoksa metot void

olarak tanımlanır. Metot içinde void anahtar sözcüğü kullanılmaz. Bu nedenle C seçeneği de doğru cevap değildir.

21. Cevap C. final olarak ifade edilen değişkenin değeri değiştirilemez. Bu nedenle parametre olarak gelen score değişkeni, score++ post-increment komutundan dolayı final olamaz. return ifadesinde de result değişkenine ekleme yapılmak istendiği için result değişkeni de final olamaz. Böylece 2 tane final ifadesinin kaldırılması kodu derlenir hale gelecektir.
22. Cevap D. Aşağıdaki bilgilere göre
    super() metot çağrısı ile üst sınıfın kurucu metodunu çağırmak için kullanılır.
    super anahtar sözcüğü ise üst sınıfın bir üyesine erişim maksatlı kullanılır.
    this() metot çağrısı ile mevcut sınıfın kurucu metodunu çağırmak için kullanılır.
    this anahtar sözcüğü ise mevcut sınıfın bir üyesine erişim maksatlı kullanılır.
    Kaynak: https://www.geeksforgeeks.org/difference-between-super-and-super-in-java-with-examples/
23. Cevap B. Verilen metodun erişim belirleyicisi package-private (default)'dir. Bu nedenle aynı paket içinde yer alan sınıflardan erişilebilir.
24. Cevap A. Protect sınıfının strength isimli değişkenini private hale getirmek, bu değişkeni sadece Protect sınıfından değiştirilebilir duruma getireceği için kapsüllemeyi geliştirir. I ifadesi doğrudur. setter ve getter eklemek kapsüllemeyi geliştirmeyeceği için II ve III ifadeleri fayda sağlamaz.
25. Cevap A. Java'da bir değişken veya metodun isminin ilk karakteri nümerik olamaz. Ayrıca Java'nın rezerve sözcükleri bir metoda ya da değişkene verilemez. İsimlendirmede "-" karakteri kullanılmaz. Bu nedenle B, C ve D seçenekleri geçersiz metot isimleridir.
    Kaynak: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/variables.html
26. Cevap D. static anahtar sözcüğü veri tipi ve metot ismi arasında yazılmıştır. Bu nedenle kod derlenmez. static, final ve erişim belirleyicisi sözcükleri veri tipinden önce yazılmalıdır.
    Kaynak: https://www.geeksforgeeks.org/final-static-variable-java/
27. Cevap B. Java, her zaman pass-by-value kullanır. Pass-by-reference kullanmaz. Nesne referansında yapılan değişikler metoda geçirilirken yansıtılmaz. Buna göre A ve C seçenekleri yanlıştır. D seçeneğindeki ifade, A'daki ifadenin boolean veri tipi için olan daha özel şeklidir ve D seçeneği de yanlıştır. Doğru cevap B'dir. Çünkü, bir nesnenin verilerinde yapılan değişikler bu nesnenin kopyası alınan yerlerde aynı olacağı için yansıtılmış olacaktır.
    Kaynak: https://www.injavawetrust.com/tag/pass-by-value/
28. Cevap C. Gift sınıfının üyesi olan contents değişkeni final olarak tanımlandığı için ilk değer verilmelidir. Ayrıca setContents() metodunda this anahtar sözcüğü ile bu değişkene değer yüklenmek istenmesi derleme hatasıdır. final bir değişkene sonradan değer verilemez. contents değişkenindeki final ifadesi silinirse sonuç A seçeneğine benzer şekilde olacaktır.
    Kaynak: https://www.geeksforgeeks.org/final-keyword-java/
29. Cevap A. Java boolean tipte bir değişkenin "is" ile başlayan değeri true/false dönen bir metodu olabilir.
    Kaynak: https://coderanch.com/t/691781/certification/JavaBeans-naming-convenctions-recap
30. Cevap C. Java'da static import kavramı Java 1.5 ile tanıtılmıştır. static import komutu C seçeneğindeki gibi yazılır. static import ifadeleri sayesinde bir sınıfın üyelerine sınıf ismi ve nesne olmaksızın erişim sağlar ve kodun okunurluğunu artırır.
    Kaynak: https://www.geeksforgeeks.org/static-import-java/
31. Cevap D. Java'da package-private erişim belirleyicisi için bir markalama yapılmaz. Herhangi bir erişim belirleyicisi belirtilmiyorsa erişim belirleyicisi default yani package-private olur.
    Kaynak: https://docs.oracle.com/javase/tutorial/java/javaOO/accesscontrol.html

32. Cevap B. Stars sınıfının bir üst sınıfı olmadığı için süper() kurucu metot çağrısı derleme hatasına yol açar. Derlenmeyi önleyen bir hata vardır.
33. Cevap A. Bir instance metodun ya da kurucu metodun tüm static değişkenlere erişimi vardır. Bu yüzden A seçeneği doğrudur. static metotlar ve static yükleme blokları instance değişkenlere işaret edemez. B seçeneği yanlıştır. C seçeneğinde verilen static initialization blokta instance değişkenine erişime izin verilir bilgisi yanlıştır. D seçeneğinde verilen, final static bir değişkenin kurucu metot içinde olabileceği bilgisi yanlıştır.
    Kaynak: https://akursat.github.io/java/java%20se/2016/02/05/13-7-static-ve-static-olmayan-bloklar.html
34. Cevap B. Kodun derlenebilmesi için short'a dönüşebilen bir veri tipi ile dönüş yapılması gerekmektedir. A, C ve D seçenekleri short tipinden daha geniş oldukları için bu seçeneklerdeki ifadeler derleme hatasını çözmez.
35. Cevap C. Metot aşırı yükleme (method overloading) aynı isme ve aynı dönüş tipine sahip ancak farklı sayıda parametre alan metotların implemente edilmesi ile uygulanır. Bu nedenle I, II ve III ifadeleri de doğrudur.
    Kaynak: https://www.baeldung.com/java-method-overload-override
36. Cevap B. monday değişkeni String türünden final static bir değişken olduğu için ilk değer verilmesi gerekmektedir. wednesday değişkeninin tipi belirtilmediği için burada da bir derleme hatası vardır. tuesday ve thursday değişkenlerinin implementasyonunda bir problem yoktur. Dolayısıyla 2 satırın çıkarılması sonucu bu kod derlenir hale gelmektedir.
37. Cevap D. main() metodundan yapılan çağrıdaki parametre alan kurucu metot tanımlı olmadığı için kod derlenmez.
38. Cevap A. public erişim belirleyicisi aynı ve farklı paketlerdeki tüm sınıfların erişimine izin verir. private erişim belirleyicisi aynı pakette yer alan sınıfların erişimlerine izin verir. Bu nedenle cevap A seçeneğidir.
39. Cevap A. main() metodunda phone nesne referansı final olarak tanımlandığı için ilk yüklemedeki değerleri koruyarak size değeri olarak "3" değerini ekrana basar.
40. Cevap B. this anahtar sözcüğü mevcut sınıfın üyelerine erişim için kullanılır. B seçeneğindeki gibi bir sınıfa erişim sağlamaz. water() metoduna erişim için diğer 3 seçenek de kullanılabilir.
41. Cevap C. Çağrıda bulunulan metodun parametreleri sırasıyla int, String ve String dizi şeklindedir. A seçeneği 4 parametreli bir metoda çağrıda bulunduğu için yanlıştır. B ve D seçeneklerindeki metot çağrıları da farklı sayıda argümana sahip olmaları nedeniyle verilen metoda çağrıda bulunan kod örnekleri değillerdir. C seçeneğindeki ifade ile verilen metoda çağrıda bulunulabilir.
42. Cevap D. static bir değişken, sınıfın tüm örnekleri tarafından erişebilirdir. Bu nedenle D seçeneği doğrudur. A seçeneğinde verilen bilgi static final değişkenlerle ilgili olduğu için bu seçenek de yanlıştır. B seçeneğinde verilen bilgi, sadece static değişkenlerin private olarak işaretlenmesiyle ilgilidir. Bu nedenle B seçeneği de yanlıştır. C seçeneği yanlıştır. Çünkü, static import ifadelerinin hem değişkenlerin hem metotların referansını kullanabilir.
43. Cevap A. Her kurucu metodun ilk satırında, this() ya da super() kurucu metot çağrıları eklenmesi gerektiği bilgisi yanlış olduğu için A seçeneği yanlıştır. Bir sınıfın kurucu metodu olmasa da derleyici parametresiz bir kurucu metodu arka planda implemente eder. Ata sınıfın kurucu metoduna parametre geçilebileceği için C seçeneğindeki bilgi de doğrudur. Bir final instance değişkeni tanımlandığı anda ya da static bir initialization blokta yüklenmelidir. Bu nedenle aranan seçenek A seçeneğidir.
44. Cevap D. Son static yükleme(initialization) bloğu, static olmayan bir instance değişkene erişebilir. Bu yüzden, kaç tane final modifier'ın kaldırıldığına bakmaksızın kodun

derlenmeyeceği söylenir. height değişkenine 4 değeri atanan satırın kaldırılması ile kod derlenecektir. Bu nedenle doğru cevap D seçeneğidir.

45. Cevap D. RainForest sınıfında tanımlanan kurucu metodun ilk satırı bir kurucu metot çağrısı olmadığı için derleyici argümansız super() kurucu metot çağrısını otomatik olarak deklare eder. Üst sınıf olan Forest sınıfı argümansız super() kurucusunu tanımlamamıştır. Bu nedenle RainForest() kurucu metodu derlenmez. Doğru cevap D seçeneğidir.
46. Cevap A. main() metodundaki kurucu metot çağrısı int parametreli metodu tetikler ve sonuç 5 olarak ekrana basılır.
47. Cevap C. getScore() metodu Long tipinden parametre beklediği halde int tipli parametre gönderildiği için m2 satırından dolayı kod derlenmez. Metodun parametresi int yapılır ve return ifade long tipine cast edilirse sonuç 8 olarak ekrana basılır.
48. Cevap A. Java'da metot isimleri harflerle, altçizgi, "$" karakterleri ile başlayabilir. Bu nedenle $sprint isminde bir metot yazılabilir.
49. Cevap B. protected erişim belirleyicisi, aynı pakette yer alan sınıflara ve alt sınıflara erişim izni sağlar. Bu nedenle, doğru cevap B seçeneğidir.
    Kaynak: https://www.baeldung.com/java-protected-access-modifier
50. Cevap D. static import ifadeleri, farklı bir sınıfın static üyelerini kullanmak için kullanılmaktadır. Bu durumda Bank sınıfındaki withDrawal() ve deposit() metotları static olarak işaretlenmez. Bank sınıfının bir örneğinin kullanılması gerekir ve static metot olarak import edilemezler. Bank sınıfındaki iki metot static olarak işaretlenseydi, Bank sınıfındaki birden fazla metodun import edilebilmesi için A seçeneğindeki gibi bir static import yazılabilirdi. B seçeneğindeki ifadede static ifadesi import ifadesinden önce yazıldığı için yazım hatalıdır. C seçeneğindeki ifade ise static import edilemeyen bir sınıfı import eder.
    Kaynak: https://www.javatpoint.com/static-import-in-java
    https://docs.oracle.com/javase/7/docs/technotes/guides/language/static-import.html