Java Metodları - Lab

# Başlık: Java Büyücüsünün Arayışı

## Konu:

Mistik bir krallıkta, yetenekli metod kullanımıyla tanınan bir Java Büyücüsü bulunmaktaydı. Bu büyücü bir çırak arıyordu ve seçilen kişi yükselen bir Java tutkunu olan Alex oldu. Alex bu yolculuğa başladığında, büyücü bir kriz öngören eski bir yazıt açığa çıkardı. Bir zamanlar huzurlu olan krallık, krallığın büyü akışını bozan kötücül bir kod yüzünden karanlıkla kaplanmıştı.

## Soru 1: Yazıtın Bulmacası

Yazıtı çözerken, Alex bir bulmaca ile karşılaştı: “Dizelerin ve seçimlerin krallığında, kodun lanetini dağıtacak metodu bul. Sırrı, belirli öğeleri tanımlayan ve yerine yeni öğeler ekleyen bir kodda yatıyor. Verilen bir dizgedeki belirli bir karakterin tüm örneklerini değiştiren bir Java metodu yaz.”

**Amaç:** Verilen bir dizgedeki belirli bir karakterin tüm örneklerini değiştiren bir metot oluşturun.

**Teknik Talimatlar:** - replaceCharacter adında üç parametre alacak bir metot tanımlayın: bir dize (inputStr), değiştirilecek bir karakter (targetChar) ve yerine geçecek bir karakter (replacementChar). - inputStr üzerinde dolaşarak tüm targetChar örneklerini replacementChar ile değiştirecek şekilde metodu uygulayın. - Değiştirilmiş dizgeyi dönün.

### Java Şablonu:

public class StringReplacer {  
 // Bir dizgedeki karakterleri değiştirmek için metodu tanımlayın  
 public static String replaceCharacter(String inputStr, char targetChar, char replacementChar) {  
 // inputStr üzerinde dolaşarak targetChar'ın tüm örneklerini replacementChar ile değiştirin  
  
 // Değiştirilmiş dizgeyi dönün  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Metodu burada test edin  
 String input = "örnek dizge"; // Dizgeyi sağlayın  
 char target = 'ö'; // Değiştirilecek hedef karakteri sağlayın  
 char replacement = 'x'; // Yerine geçecek karakteri sağlayın  
  
 String modifiedString = replaceCharacter(input, target, replacement);  
 System.out.println("Değiştirilmiş Dizge: " + modifiedString);  
 }  
}

### Örnek Çıktı:

Değiştirilmiş Dizge: xrnek dizge

## Hikaye Gelişimi:

public class StringReplacer {  
 // Bir dizgedeki karakterleri değiştirmek için metodu tanımlayın  
 public static String replaceCharacter(String inputStr, char targetChar, char replacementChar) {  
 String modifiedString = "";  
  
 for (int i = 0; i < inputStr.length(); i++) {  
 char currentChar = inputStr.charAt(i);  
 if (currentChar == targetChar) {  
 modifiedString += replacementChar;  
 } else {  
 modifiedString += currentChar;  
 }  
 }  
  
 return modifiedString;  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Metodu burada test edin  
 String input = "örnek dizge"; // Dizgeyi sağlayın  
 char target = 'ö'; // Değiştirilecek hedef karakteri sağlayın  
 char replacement = 'x'; // Yerine geçecek karakteri sağlayın  
  
 String modifiedString = replaceCharacter(input, target, replacement);  
 System.out.println("Değiştirilmiş Dizge: " + modifiedString);  
 }  
}

Kararlılıkla, Alex Java bilgisine daldı. Düşünerek ve deneyerek, Alex kötücül karakterin tüm örneklerini değiştire

bilen bir metot geliştirdi. Büyücü onaylayarak başını salladı ve lanet kalkmaya başladığında, krallıkta umut ışığı tekrar belirdi.

## Soru 2: Eşyaların Şifresi

Krallıkta dolaşırken, Alex ve Java Büyücüsü, eski bilgilerin özünü taşıyan büyülü eşyalarla karşılaştı. Bu sırların kilidini açmak için, Büyücü, bu eşyaların büyülerini çözecek bir dizi bağlantılı metodun oluşturulması gerektiğini ortaya çıkardı. Her bir metot, öncekinin çıktısına dayanıyordu; bu da eşyaların gizemlerini çözmek için hayati öneme sahip bir bilgi zinciri oluşturdu.

**Amaç:** Eşyaların özünü belirli dönüşümler yaparak çözmek için bir dizi metot oluşturun.

**Teknik Talimatlar:** - extractEssence, purifyEssence ve decodeArtifact adlarında üç metot tanımlayın. - extractEssence, artifact adında bir dize parametresi alacak ve özü çıkarılmış bir dize döndürecek. - purifyEssence, çıkarılan özü alacak ve bunu arındırılmış bir sürüm olarak döndürecek. - decodeArtifact, purifyEssence çıktısını kullanacak ve nihai dönüşümü yaparak çözülmüş eşyayı döndürecek.

### Java Şablonu:

public class ArtifactDecoder {  
 // Eşyadan özü çıkarmak için metodu tanımlayın  
 public static String extractEssence(String artifact) {  
 // Eşyadan özü çıkarmak için mantığı uygulayın  
 // Örnek: ilk ve son 9 karakteri kaldırarak özü çıkarma  
 // Özü çıkarılmış dizeyi dönün  
 return ""; // Yer tutucu dönüş ifadesi  
 }  
  
 // Çıkarılan özü arındırmak için metodu tanımlayın  
 public static String purifyEssence(String extractedEssence) {  
 // Çıkarılan özü arındırmak için mantığı uygulayın (örnek: küçük harfe dönüştürme)  
 // Arındırılmış özü dönün  
 return ""; // Yer tutucu dönüş ifadesi  
 }  
  
 // Arındırılmış özü kullanarak eşyayı çözmek için metodu tanımlayın  
 public static String decodeArtifact(String purifiedEssence) {  
 // Arındırılmış özü kullanarak eşyayı çözmek için mantığı uygulayın (örnek: dizeyi tersine çevirme)  
 // Çözülmüş eşyayı döndürün  
 return ""; // Yer tutucu dönüş ifadesi  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Metotları bir sırayla çağırarak test edin  
 String artifact = "Büyülü eşya dizesi"; // Büyülü eşyayı sağlayın  
  
 String extractedEssence = extractEssence(artifact);  
 String purifiedEssence = purifyEssence(extractedEssence);  
 String decodedArtifact = decodeArtifact(purifiedEssence);  
  
 System.out.println("Çözülmüş Eşya: " + decodedArtifact);  
 }  
}

### Örnek Çıktı:

Çözülmüş Eşya: ay

## Hikaye Gelişimi:

public class ArtifactDecoder {  
 // Eşyadan özü çıkarmak için metodu tanımlayın  
 public static String extractEssence(String artifact) {  
 // Eşyadan özü çıkarmak için mantığı uygulayın  
 return artifact.substring(9, artifact.length() - 7); // Örnek: ilk ve son 9 karakteri kaldırarak özü çıkarma  
 }  
  
 // Çıkarılan özü arındırmak için metodu tanımlayın  
 public static String purifyEssence(String extractedEssence) {  
 // Çıkarılan özü arındırmak için mantığı uygulayın (örnek: küçük harfe dönüştürme)  
 return extractedEssence.toLowerCase();  
 }  
  
 // Arındırılmış özü kullanarak eşyayı çözmek için metodu tanımlayın  
 public static String decodeArtifact(String purifiedEssence) {  
 // Arındırılmış özü kullanarak eşyayı çözmek için mantığı uygulayın (örnek: dizeyi tersine çevirme)  
 return new StringBuilder(purifiedEssence).reverse().toString();  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Metotları bir sırayla çağırarak test edin  
 String artifact = "Büyülü eşya dizesi"; // Büyülü eşyayı sağlayın  
  
 String extractedEssence = extractEssence(artifact);  
 String purifiedEssence = purifyEssence(extractedEssence);  
 String decodedArtifact = decodeArtifact(purifiedEssence);  
  
 System.out.println("Çözülmüş Eşya: " + decodedArtifact);  
 }  
}

Alex metodları oluştururken, extractEssence, purifyEssence ve decodeArtifact arasındaki karmaşık bağlantıyı fark etti. extractEssence metodu eşyanın özünü izole etti ve bunu purifyEssence’a besledi, bu da bunu dikkatlice temizledi. Son olarak, decodeArtifact, arıtılmış özü kullanarak eşyayı çözerek eşyanın içinde gizlenmiş olan şifreli bilgeliği açığa çıkardı. Her metotun çıktısı sorunsuz bir şekilde bir sonrakine akarken, büyülü eşyaların içindeki uzun süredir gizlenmiş olan bilgi ortaya çıktı.

## Soru 3: Yaratıkların Büyüsü

Alex, Java Büyücüsü ile yolculuğuna devam ettikçe, garip bir anormalliğe maruz kalan efsanevi yaratıklarla karşılaştı. Bu yaratıklar, bir zamanlar barışçılken, büyü işleyişindeki bozulmadan dolayı saldırgan hale gelmişlerdi. Büyücü, yalnızca belirli bir metotun bu büyüleri karşılayabileceğini ve yaratıkların huzurunu geri getirebileceğini ortaya çıkardı. İlerledikçe, farklı büyü türlerine ve savunmasızlıklarına sahip çeşitli yaratıklarla karşılaştılar.

**Amaç:** Büyü türüne ve çarpanına dayalı olarak hasar hesaplamak için bir metot oluşturun.

**Teknik Talimatlar:** - calculateDamage adında iki parametre alacak bir metot tanımlayın: spellType (bir dize) ve multiplier (bir double). - Farklı spellType durumlarını değerlendirmek için koşullu ifadeler kullanın ve verilen multiplier’a dayanarak hasarı hesaplayın. - Hesaplanan hasarı bir tamsayı olarak döndürün.

### Java Şablonu:

public class SpellDamageCalculator {  
 // Büyü türüne ve çarpanına dayalı olarak hasarı hesaplamak için metodu tanımlayın  
 public static int calculateDamage(String spellType, double multiplier) {  
 // Koşullu ifadeler kullanarak spellType'a göre hasarı hesaplayın  
  
 // Hesaplanan hasarı bir tamsayı olarak döndürün  
 return 0; // Yer tutucu dönüş ifadesi  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Farklı parametrelerle metodu birden çok kez çağırarak burada test edin  
 int damage1 = calculateDamage("Ateş", 1.5); // 1.5 çarpanıyla Ateş büyüsü için hasarı hesaplayın  
 System.out.println("Hasar 1: " + damage1);  
  
 int damage2 = calculateDamage("Buz", 1.2); // 1.2 çarpanıyla Buz büyüsü için hasarı hesaplayın  
 System.out.println("Hasar 2: " + damage2);  
  
 // Diğer büyü türleri ve çarpanları için hasarı hesaplayın  
 // ...  
 }  
}

### Örnek Çıktı:

Hasar 1: 15  
Hasar 2: 12

## Hikaye Gelişimi:

public class SpellDamageCalculator {  
 // Büyü türüne ve çarpanına dayalı olarak hasarı hesaplamak için metodu tanımlayın  
 public static int calculateDamage(String spellType, double multiplier) {  
 int baseDamage = 10; // Tüm büyüler için bir taban hasar değeri  
  
 // Koşullu ifadeler kullanarak spellType'a göre hasarı hesaplayın  
 if (spellType.equals("Ateş")) {  
 return (int) (baseDamage \* multiplier); // Ateş büyüsü hasarı hesaplama  
 } else if (spellType.equals("Buz")) {  
 return (int) (baseDamage \* multiplier); // Buz büyüsü hasarı hesaplama  
 } else {  
 // Diğer büyü türleri için hasarı ele alın veya varsayılan bir hasar değeri sağlayın  
 return 0; // Diğer büyüler için işleme koyma yeri  
 }  
 }  
  
 // Test için ana metot  
 public static void main(String[] args) {  
 // Farklı parametrelerle metodu birden çok kez çağırarak burada test edin  
 int damage1 = calculateDamage("Ateş", 1.5); // 1.5 çarpanıyla Ateş büyüsü için hasarı hesaplayın  
 System.out.println("Hasar 1: " + damage1);  
  
 int damage2 = calculateDamage("Buz", 1.2); // 1.2 çarpanıyla Buz büyüsü için hasarı hesaplayın  
 System.out.println("Hasar 2: " + damage2);  
  
 // Diğer büyü türleri ve çarpanları için hasarı hesaplayın  
 // ...  
 }  
}

Alex, calculateDamage metodunu uygularken, çeşitli büyü yapan yaratıklarla karşılaştı. Bu metodu kullanarak, 1.5 çarpanlı Ateş büyüsünün verdiği hasarı başarılı bir şekilde hesapladı ve ateşe duyarlı yaratıklara yıkıcı darbeler vurdu. Ayrıca, 1.2 çarpanlı Buz büyüsünün etkisini buzla uyumlu yaratıklar üzerinde çözebildi. Metodun her başarılı kullanımı, metodların farklı durumlarda tanımlanmasının gücünü ve farklı durumlarda yeniden kullanılabilirliğini gösterdi, Java’nın metod uygulamasının çeşitliliğini ve verim

liliğini ortaya koydu.

Alex’in Java metodlarındaki uzmanlığı arttıkça, krallıktaki umut ışığı da büyüdü. Büyücü, çırakların dikkate değer ilerlemesini kabul ederek gururla gülümsedi. Her bir meydan okuma aşıldıkça, karanlık kayboldu ve krallık içinde barış ve büyü yeniden restore edilmeye başladı.