**Sayfa 44**

Keşif testi

- Keşif Testi, test ekibi tarafından gerçekleştirilen gayri resmi testlerdir. Bu testin amacı,

  uygulama ve uygulamada var olan kusurları aramak. Bazen bu test sırasında büyük

  keşfedilen kusur sistem arızasına bile neden olabilir.
- Keşif testi sırasında, hangi akışı test ettiğinizi ve daha önce hangi aktiviteyi yaptığınızı takip etmeniz önerilir.

  belirli akışın başlangıcı.

**Testlerinizi nasıl otomatikleştirebilirsiniz?**

- Bir kişi yukarıda bahsedilen tüm testleri uygulayabilir, ancak bunu yapmak çok pahalı ve verimsiz olacaktır. Gibi

  insanlar, çok sayıda eylemi tekrarlanabilir ve güvenilir bir şekilde gerçekleştirme kapasitemiz sınırlıdır. Ama bir makine

  bunu kolayca yapabilir ve giriş / şifre kombinasyonunun 100. kez şikayet etmeden çalıştığını test eder.
- Testlerinizi otomatikleştirmek için, önce testlerinizi kendinize uygun bir test çerçevesi kullanarak programlı olarak yazmanız gerekir.

  uygulama. PHPUnit , Mocha , RSpec , PHP, JavaScript ve Ruby için kullanabileceğiniz test çerçeveleri örnekleridir.

  sırasıyla. Her dil için birçok seçenek var, bu yüzden biraz araştırma yapıp sormanız gerekebilir

  sizin için en iyi çerçevenin ne olacağını öğrenmek için geliştirici toplulukları.
- Testleriniz terminalinizden komut dosyası aracılığıyla yürütüldüğünde, bunların bir

  Bamboo gibi sürekli entegrasyon sunucusu veya Bitbucket Pipelines gibi bir bulut hizmeti kullanın. Bu araçlar



**Sayfa 45**

# JAVA

**1. Java Sanal Makinesi**

- JVM, derlenmiş java sınıfı dosyalar için bir çalışma zamanı ortamı olan Java Sanal Makinesi anlamına gelir.

**2. JavaScript ve Java aynı mı?**

- Java bir OOP programlama dilidir, Java Script ise bir OOP kodlama dilidir.
- Java, JavaScript kodu yalnızca bir tarayıcıda çalıştırılırken sanal bir makinede veya tarayıcıda çalışan uygulamalar oluşturur.
- JavaScript kodunun tamamı metin içindeyken Java kodunun derlenmesi gerekir.
- Farklı eklentiler gerektirirler.

**3. Java Runtime Environment**

- JRE, bir Java programını çalıştırmak için ihtiyacımız olan şeydir ve JVM'nin çalışma zamanında kullandığı kitaplıklar ve diğer dosyaları içerir.

• JRE = JVM + Kitaplık Sınıfları

**4. Java Geliştirme Kiti**

• JDK, Java kaynak kodunu derlemek için ihtiyacımız olan şeydir ve JRE, geliştirme araçlarını içerir.

• JDK = JRE + Geliştirme araçları

**5. Java Interview için popüler konular nelerdir?**

Java röportajı için popüler konulardan bazıları şunlardır:

• OOPS Kavramları
• Java Dizesi
• Koleksiyon Çerçevesi
• Çoklu kullanım
• Jenerikler
• İstisna işleme
• Akış API'si
• Lambda İfadeleri
• En Son Sürüm Özellikleri
• Java EE Frameworks - Spring, Hibernate vb.

**6. Gelişmiş Java Konuları nelerdir?**

Java röportajı için popüler konulardan bazıları şunlardır:

• Yığın ve Yığın Bellek
• Çöp toplama
• Reflection API
• İş Parçacığı Kilitlenme
• Java ClassLoader
• Java Günlük API'si
• Java'da uluslararasılaşma
• Java Modül Sistemi

**7. Tüm sınıfların üst sınıfı hangi sınıftır?**

• java.lang.Object , tüm java sınıfları için kök sınıftır ve onu genişletmemize gerek yoktur.

**8. Yöntem nedir?**

• Bir işlemi gerçekleştirmek için bir arada gruplandırılmış ifadelerin toplanması. **System.out.println () 'i** aradığınızda

yöntem, örneğin, sistem konsolda bir mesaj görüntülemek için gerçekte birkaç deyimi yürütür.

• Bir yöntem, adla atıfta bulunulan ve bir programın herhangi bir noktasında yalnızca aşağıdaki yöntemlerle çağrılabilen (çağrılan) bir kod kümesidir.

yöntemin adını kullanarak. Bir yöntemi, verilere göre hareket eden ve genellikle bir değer döndüren bir alt program olarak düşünün. Her yöntem kendi adı var.

**9. Yapıcı nedir?**

• Java'daki bir kurucu, nesneleri başlatmak için kullanılan özel bir yöntemdir. Yapıcı, bir nesnenin bir nesnesi olduğunda çağrılır.

sınıf oluşturulur.

• new () anahtar sözcüğü kullanılarak bir nesne her oluşturulduğunda, en az bir yapıcı (varsayılan kurucu olabilir) için çağrılır

aynı sınıfın veri üyelerine başlangıç değerleri atayın.



## Sayfa 46

**10. Oluşturucu ve Yöntem arasındaki fark?**

• Oluşturucunun bir dönüş türü yoktur ve kurucunun adı, sınıf adıyla aynı olmalıdır.

o Oluşturucu, yeni bir nesne oluşturulduğunda otomatik olarak çağrılır. Oluşturucu örtük olarak çağrılır.
o Herhangi bir kurucumuz yoksa Java derleyicisi varsayılan bir kurucu sağlar.
o Oluşturucular alt sınıflar tarafından miras alınmaz

• Yöntemin bir dönüşü vardır ve yöntemin adı sınıf adıyla aynı olabilir veya olmayabilir

o Yöntem açıkça çağrılır.
o Yöntem hiçbir durumda derleyici tarafından sağlanmamaktadır.
o Yöntemler alt sınıflar tarafından miras alınır.

**11. Yerel bir değişken ile bir durum değişkeni arasındaki fark nedir?**

• **Yerel bir değişken** tipik olarak bir yöntem, yapıcı veya blok içinde kullanılır ve yalnızca yerel kapsama sahiptir. Böylece bu değişken,

yalnızca bir blok kapsamında kullanılabilir.

• Yerel bir değişkene sahip olmanın en iyi yararı, sınıftaki diğer yöntemlerin o değişkenden haberdar olmayacak olmasıdır.

*Misal*

```
eğer (x> 100) {
        Dize testi = "Alberto";
}
```

• Java'daki bir **örnek değişkeni** , nesnesinin kendisine bağlı bir değişkendir. Bu değişkenler bir

sınıf, ancak bir yöntemin dışında. Bu sınıfın her nesnesi, onu kullanırken değişkenin kendi kopyasını oluşturacaktır. Böylece herhangi biri Değişkende yapılan değişiklikler, o sınıfın başka herhangi bir örneğini yansıtmayacak ve o özelliğe bağlı olacaktır.
yalnızca örnek.
*Misal*

```
class Test {
        public String EmpName;
        public int empAge;
}
```

**12. Nesneye Yönelik Programlama (OOP)**

• OOP, eylemler (mantık ve işlevler) yerine nesne etrafında düzenlenen bir programlama dili modelidir.

• Diğer bir deyişle, OOP esas olarak mantık yerine manipüle edilmesi gereken nesnelere odaklanır. Bu yaklaşım programlar için ideal, büyük ve karmaşık kodlar ve aktif olarak güncellenmesi veya bakımı yapılması gereken;
• Geliştirme ve bakımı kolaylaştırır - Veri gizleme sağlar - Gerçek dünyayı simüle etme yeteneği sağlar.

**OOP dili 4 prensibi takip eder** :

o **Kapsülleme** : Özel anahtar kullanarak verilere doğrudan erişimi gizleyebiliriz ve alıcı kullanarak özel verilere erişebiliriz ve ayarlayıcı yöntemi.

o **Soyutlama** : Uygulama detaylarının gizlenmesi ve kullanıcıya sadece işlevselliğin gösterilmesi işlemidir. Soyutlama nasıl yaptığı yerine nesnenin ne yaptığına odaklanmanızı sağlar.

o **Kalıtım** : İki sınıf arasındaki ilişkiyi tanımlamak için kullanılır. Bir çocuk sınıfı tüm özellikleri edindiğinde ve miras olarak bilinen ebeveyn sınıfın davranışları. Çocuk sınıfı, ebeveyn sınıfında yazılan tüm kodları yeniden kullanabilir. Sağlar kodun yeniden kullanılabilirliği.

o **Çok biçimlilik** : Nesnenin çoklu biçimde davranma yeteneğidir. Java'da polimorfizmin en yaygın kullanımı değişkenin bir üst sınıf başvuru türü, bir alt sınıf nesnesine başvurmak için kullanıldığı zamandır.

*Misal*

WebDriver sürücüsü = yeni ChromeDriver ();

Polimorfizme ulaşmak için yöntem aşırı yükleme ve geçersiz kılma kullanıyoruz.



---

## Sayfa 47

**13. Kapsülleme nedir ve nasıl kullandınız?**

• Değişkenleri özel yaparak ve genel alıcı ve ayarlayıcı yöntemleri sağlayarak veri gizleme.
• Projemde, test verilerini ve gerçek verileri yönetmek için birden fazla POJO / BEAN sınıfı oluşturdum.
• EX: JSON'u API yanıtından alıp POJO sınıfımın nesnesine dönüştürüyorum, tüm değişkenler alıcılarla özeldir ve ayarlayıcı.

**14. Soyutlama kavramı nedir ?**

• OOP'de, soyutlama, uygulama ayrıntılarını kullanıcıdan gizleme sürecidir, yalnızca işlevsellik sağlanacaktır. kullanıcıya.
• Başka bir deyişle, kullanıcı nesnenin nasıl yaptığı yerine ne yaptığı bilgisine sahip olacaktır.
• Java'da soyutlama, Soyut sınıflar ve arayüzler kullanılarak gerçekleştirilir.
• Örneğin: banka hesabınızda çevrimiçi oturum açtığınızda, kullanıcı kimliğinizi ve parolanızı girip oturum açma tuşuna basarsınız. Ne daha sonra, sunucuya gönderilen giriş verilerinin nasıl doğrulandığı, tümüyle sizden soyutlanır.

**15. Soyutlama ve Kapsülleme Arasındaki Fark ?**

• **Soyutlama** , nesnenin **nasıl** yaptığı yerine **ne** yaptığına odaklanmanızı sağlar .
o **Kapsülleme** , nesnenin bir şeyi nasıl yaptığına dair dahili ayrıntıları gizlemek anlamına gelir.
• **İstenmeyen** verileri gizlemek ve ilgili verileri vermek için **soyutlama** kullanılır .
o **Kapsülleme** , kodu ve verileri gizlemek ve verileri dışarıdan korumak anlamına gelir.
• **Soyutlama** , Soyut sınıf ve Arayüzler kullanılarak elde edilebilir
o **Kapsülleme** , "özel" anahtar kelime kullanılarak elde edilebilir.

**16. Soyut Sınıf ve Arayüz arasındaki fark ?**

• Temel fark, bir Java arayüzünün yöntemlerinin dolaylı olarak soyut olması ve uygulamalarının olmamasıdır. Java özeti sınıf, varsayılan bir davranışı uygulayan örnek yöntemlerine sahip olabilir.
• Abstract anahtar sözcüğüyle bildirilen bir sınıf, soyut sınıf olarak bilinir. Soyut ve soyut olmayan yöntemlere sahip olabilir.
• Bir Arayüz, bir sınıfın taslağıdır. Bu bir şablondur ve interface anahtar sözcüğü ile bildirilmiştir. Soyut yöntemlere sahip olabilir, varsayılan yöntemler, statik yöntemler ve genel nihai statik değişkenler
• Abstract sınıfını kullanmak istediğimizde, " **extended** " anahtar sözcüğünü kullanırız. Arayüzü kullanmak istediğimizde, " uygulama " kullanıyoruz anahtar kelime.
• Soyut sınıf ve arayüz, soyutlama elde etmek için kullanılır Her ikisi de somutlaştırılamaz; bir nesne yaratamayız.

**17. Polimorfizm nedir?**

• Polimorfizm, OOP'de çok önemli bir kavramdır çünkü;
o çağrının yapıldığı nesneye bağlı olarak çalışma süresindeki uygulamaların davranışını değiştirmeyi sağlar. olur.
o Polimorfizm ile; bir nesnenin farklı biçimleri olabilir
• İki tip → Statik olan **Derleme Zamanı** ve alt ve üst sınıf ile ilgili olan **Çalışma Zamanı** Polimorfizmi.
• Çok biçimlilik, Yöntem aşırı yükleme ve yöntemi geçersiz kılma kavramı kullanılarak gerçekleştirilir. Bu sadece olabilir sınıflar, miras kullanarak üst ve alt ilişki altında olduğunda.

**18. Kalıtım nedir?**

• Kalıtım, ebeveyn-çocuk ilişkisi olarak da bilinen **IS-A** ilişkisini temsil eder .
• Java'da bir sınıfın başka bir sınıfın özelliklerini (alanları ve yöntemleri) devralmasına izin verilen mekanizmadır.
• Java'da kalıtımın arkasındaki fikir, mevcut sınıflar üzerine inşa edilmiş yeni sınıflar oluşturabilmenizdir.
• Var olan bir sınıftan miras aldığınızda, üst sınıfın yöntemlerini ve alanlarını yeniden kullanabilirsiniz.

• Ayrıca, mevcut sınıfınıza yeni yöntemler ve alanlar da ekleyebilirsiniz.



## Sayfa 48

• Kodun yeniden kullanımı, kalıtımın en önemli yararıdır çünkü alt sınıflar,

süper sınıf.

**19. Önemli terminoloji Kalıtım ?**

• **Sınıf:** ortak özelliklere sahip nesneler grubu. Nesnelerin oluşturulduğu bir şablon veya plandır.

• **SuperClass** : miras alınan sınıf (veya bir temel sınıf veya bir üst sınıf).

• **Alt** Sınıf: başka bir sınıftan (veya türetilmiş bir sınıftan, genişletilmiş sınıftan veya alt sınıftan) miras alan sınıf.

o Alt sınıf, üst sınıf alanlara ve yöntemlere ek olarak kendi alanlarını ve yöntemlerini ekleyebilir.

• **Yeniden kullanılabilirlik** : bir sınıf oluşturduğunuzda mevcut sınıfın alanlarını ve yöntemlerini yeniden kullanmanızı kolaylaştıran bir mekanizma.

yeni sınıf. Önceki sınıfta zaten tanımlanmış olan aynı alanları ve yöntemleri kullanabilirsiniz.

**20. Çok Biçimlilik ve Kalıtım Arasındaki Fark**

• Gerçek dünyada olduğu gibi, Miras, iki sınıf arasındaki ilişkiyi tanımlamak için kullanılır. Baba-Oğul'a benzer

ilişki. Java'da, Ebeveyn sınıfımız (süper sınıf olarak da bilinir) ve alt sınıfımız (alt sınıf olarak da bilinir) vardır. Benzer

gerçek dünyada Çocuk, Ebeveynlerin niteliklerini, yöntemlerini ve kodlarını miras alır.

o Bir alt sınıf, Parent sınıfında yazılan tüm kodları yeniden kullanabilir ve yalnızca

Ebeveyn.

o Kalıtım aslında kodun yeniden kullanılması içindir.

• Öte yandan, Polimorfizm, nesnenin çoklu biçimde davranma yeteneğidir.

o Aşırı yükleme ve geçersiz kılma olarak sınıflandırılır.

• Bu arada, aslında birbirleriyle ilişkilidirler, çünkü Polimorfizmi mümkün kılan mirası,

iki sınıf arasındaki herhangi bir ilişki. Polimorfik kod yazmak mümkün değildir.

o Dinamik Polimorfizm → Geçersiz Kılma

o Statik Polimorfizm → Aşırı Yükleme

**21. Metot Aşırı Yüklemesi ile Metodu Geçersiz Kılma arasındaki fark ?**

• Aşırı yükleme ve geçersiz kılma arasındaki ilk ve en önemli fark şudur:

o aşırı yükleme durumunda, yöntem adı aynı olmalı, ancak parametreler farklı olmalıdır;

o geçersiz kılma durumunda, yöntem adı ve parametreleri aynı olmalıdır

• Metot aşırı yükleme ile geçersiz kılma arasındaki ikinci büyük fark şudur;

o Metodu aynı sınıfta aşırı yükleyebiliriz ancak metodu geçersiz kılma kalıtımı olan iki sınıfta gerçekleşir.

ilişki.

• Java'da static, final ve private metodu geçersiz kılamayız, ancak Java'da static, final ve private metodu aşırı yükleyebiliriz.

• Yöntem aşırı yüklemesinde dönüş tipi aynı veya farklı olabilir. Yöntemin geçersiz kılınmasında, dönüş türü aynı veya eşdeğişken olmalıdır

yazın.

**22. Değişmez nedir?**

• Değişmez, bir nesnenin yapıcısı yürütmeyi tamamladığında, örneğin değiştirilemeyeceği anlamına gelir.

• Bu, başka birinin gideceğinden endişe etmeden, etrafınızdaki nesneye referansları iletebileceğiniz anlamına geldiği için kullanışlıdır.

içeriğini değiştirin.

Özellikle eşzamanlılıkla uğraşırken, asla değişmeyen nesnelerle ilgili kilitleme sorunları yoktur.

```
class Foo {
    özel final String myvar ;
    public Foo (final String initialValue) {
    bu . myvar = initialValue ;
}
    Genel Dize getValue () {
    bunu geri ver . myvar ;
    }
```



## Sayfa 49

**23. Dinamik / çalışma zamanı bağlamasına karşı statik bağlama nedir?**

• Statik bağlama, aşırı yükleme ve dinamik bağlama, yöntem aşırı yüklemesidir

**24. Erişim değiştirici nedir ve farklı erişim değiştiriciler nelerdir?**

• Java, sınıflar, değişkenler, yöntemler ve yapıcılar için erişim düzeylerini ayarlamak için bir dizi erişim değiştirici sağlar.

o Paket tarafından görülebilir, varsayılan. Değiştiriciye gerek yoktur.

o Yalnızca sınıfa görünür (özel).

o Dünyaya görünür (halka açık).

o Paket ve tüm alt sınıflar (korumalı) tarafından görülebilir.

**25. Java'da Genel, Özel ve Korumalı değiştirici arasındaki fark nedir?**

- Java'da sınıf, yöntem ve değişkenlerin erişilebilirliğini belirten erişim değiştiricisi. İçinde dört erişim değiştirici vardır

  Java, yani Genel, Özel, Korumalı ve Varsayılan.
- Bu erişim modifikasyonları arasındaki fark şudur;
  - o En önemlisi erişilebilirlik düzeyidir.
  - o Herkese her yerden erişilebilir
  - o Özel, yalnızca beyan edilen aynı sınıfta erişilebilir
  - o Varsayılana yalnızca aynı paket içinde erişilebilir
  - o Korumalıya aynı paket içinde ve aynı zamanda paketin dışında ancak sadece alt sınıflar erişilebilir.
- Üst düzey bir sınıfla özel veya korumalı değiştirici kullanamayız.
- Ayrıca, erişim değiştiricinin genel, özel veya Java'da korunan yerel değişkenler için uygulanamayacağını unutmamalıyız.

**26. Java'da Set, List ve Map arasındaki fark nedir?**

- Set, List ve Map, Java toplama çerçevesinin 3 önemli arabirimidir.
  - o Liste, *çoğaltma içerebilecek sıralı* ve indekslenmiş koleksiyon sağlar .
  - o Set, benzersiz nesnelerin *sırasız bir şekilde* toplanmasını sağlar . Set *, kopyalamaya izin vermiyor* . Liste ve Küme her ikisi de genişletilir

    koleksiyon arayüzü.
  - o Harita, Anahtar Değerine dayalı bir veri yapısı sağlar. Anahtar her zaman benzersizdir, değer çift olabilir.

**27. Liste, Set ve Harita ne zaman kullanılır?**

- İndeks kullanarak öğelere sık sık erişmemiz gerekirse, List, gitmenin bir yoludur ArrayList, index ile daha hızlı erişim sağlar.
- Elemanları saklamak istiyorsak ve bir sırayı korumalarını istiyorsak, Liste sıralı bir koleksiyondur ve sırayı korur.
- Herhangi bir Set uygulamasını seçmektense, yinelemesiz benzersiz öğeler koleksiyonu oluşturmak istiyorsak. (HashSet ...)
- Verileri Anahtar ve Değer biçiminde depolamak istiyorsak, gitmenin yolu budur. HashMap, Hashtable arasından seçim yapabiliriz ...

**28. Dizi nedir?**

- Dizi, tek bir türden sabit sayıda değeri tutan bir kap nesnesidir. Bir dizinin uzunluğu belirlenir

  dizi oluşturulduğunda. Oluşturulduktan sonra uzunluğu sabittir. Halihazırda ana dizinin bir örneğini gördünüz.

  "Merhaba Dünya!" uygulama. Bu bölümde diziler daha ayrıntılı olarak tartışılmaktadır.
- Bir dizideki her öğeye öğe adı verilir ve her öğeye sayısal dizini ile erişilir. Yukarıda gösterildiği gibi

  örnekleme, numaralandırma 0 ile başlar. Bu nedenle, örneğin 9. öğeye indeks 8'den erişilebilir.
- **Java Dizisinin Avantajı**
  - o Kod Optimizasyonu: Kodu optimize eder, verileri kolayca alabilir veya sıralayabiliriz.
  - o Rastgele erişim: Herhangi bir indeks konumunda bulunan herhangi bir veriyi alabiliriz.
- **Java Dizisinin Dezavantajı**
  - o Boyut Sınırı: Dizide yalnızca sabit boyuttaki öğeleri saklayabiliriz. Çalışma zamanında boyutunu büyütmez. Bunu çözmek için

    problem, java'da koleksiyon çerçevesi kullanılıyor.



## Sayfa 50

**29. ArrayList'in yineleme içerip içermediğini nasıl anlarsınız?**

- Kullanılabilecek birkaç yol vardır. En kısa olanı .stream (). Differt (). Count () yöntemidir

  list.size ()! = list.stream (). farklı (). count ()
- Diğer yöntemler:

```
// YÖNTEM 1
public static <T> boolean containsUnique ( Liste <T> listesi) { Set <T> set = new HashSet <> ();
return list.stream (). allMatch (t -> set.add (t));
}

// YÖNTEM 2
public static <T> boolean containsUnique ( List <T> listesi) { return list.stream (). allMatch ( yeni
HashSet <> () :: ekle);
} // sadece saf akışları işleyebildiği için değil, aynı zamanda durduğu için de en iyisi gibi görünüyor
ilk kopya (# 1 ve # 2 her zaman sonuna kadar yinelenirken)

// YÖNTEM 3
public static <T> boolean containsUnique ( Liste <T> listesi) {
        Takım <T> grubu = Yeni HashSet <> ();
        for (T t: list) {
                eğer (! set.add (t))
yanlış dönüş ; }
```

**30. Java'da Diziler ve ArrayList arasındaki fark nedir?**

- Dizi, temel Java programlamasının bir parçasıdır ve özel sözdizimine sahiptir ArrayList, koleksiyon çerçevesinin bir parçasıdır ve uygular

  Liste arayüzü
- En büyük fark şudur; Dizi sabit uzunlukta bir veri yapısıdır, bu nedenle oluşturulan Array uzunluğunu değiştirebiliriz, ArrayList

yeniden boyutlandırılabilir.

- Diğeri ise Array'in hem ilkelleri hem de nesneleri içerebilmesidir. ArrayList yalnızca nesneleri içerebilir. Olamaz ilkel türleri içerir.
- Ayrıca, Array uzunluğunun veya ArrayList boyutunun nasıl hesaplanacağı konusunda Array ve ArrayList'i karşılaştırabiliriz. Bir uzunluk için kullanıyoruz Array, bir ArrayList için size () yöntemini kullanıyoruz.

**31. İş parçacığı güvenli veya Senkronize nedir?**

- İş parçacığı güvenliği çok önemlidir ve programımızı çok iş parçacıklı ortamda güvenli hale getirme sürecidir, program iş parçacığımızı güvenli hale getirmenin farklı yolları vardır.
- **Senkronizasyon** , iplik güvenliği için en kolay ve en yaygın kullanılan araçtır.



## Sayfa 51

- JVM, senkronize kodun bir seferde yalnızca bir iş parçacığı tarafından yürütüleceğini garanti eder.
- **Senkronize edilmiş** JAVA anahtar sözcüğü, **senkronize** edilmiş kod oluşturmak için kullanılır ve dahili olarak nesne veya Sınıf üzerindeki kilitleri kullanır. sadece bir iş parçacığının eşitlenmiş kodu çalıştırdığından emin olun.
- Java senkronizasyonunun kaynağın kilitlenmesi ve kilidinin açılması üzerinde çalıştığını, bu nedenle senkronize edilen iş parçacığına girmediğini söylüyorum kodu.
- Senkronize edilmiş anahtar kelimeyi iki şekilde kullanabiliriz, biri tam bir yöntemi senkronize etmek, diğeri ise senkronize blok oluşturun.

**32. Oluşturduğunuz bir nesneyi nasıl sınıflandırırsınız?**

- Sıralayabilecektir.
- Ayrıca, nesnelerimi bir TreeSet veya TreeMap'e depolayabilirim → Örn: SONRAKİ SAYFA
- Java, bir listeyi sıralamak için birkaç yol sağlar.
  - o KARŞILAŞTIRILABİLİR - KARŞILAŞTIRICI arayüzleri sıralama için kullanılabilir. Bu durumlarda, CompareTo öğesini geçersiz kılmalıyız yöntem.
- Diğer bir yol, bir karşılaştırıcı kullanabilen Liste arabirimi sıralama yöntemidir. Bu yöntemle artan veya artan şekilde sıralayabiliriz Azalan.

```
users.sort (Karşılaştırıcı.comparing (Kullanıcı :: getUserID));
```

- Orijinal listeyi değiştirmek istemiyorsak, ancak yeni bir sıralanmış liste döndürmek istiyorsak; daha sonra sıralı () yöntemini kullanabiliriz. Akış arayüzü…

```
List <Kullanıcı> sıralanmışUsers = users.stream ()
  .sorted (Karşılaştırıcı.comparing (Kullanıcı :: getUserID))
  .collect (Collectors .toList ());
```

**33. Java'da Hashtable ve HashMap arasındaki fark nedir?**

Java'da HashMap ve Hashtable arasında birkaç fark vardır:

- Hashtable senkronize edilirken HashMap senkronize değildir. Bu, HashMap'i iş parçacıklı olmayan uygulamalar için daha iyi hale getirir. senkronize edilmemiş Nesneler tipik olarak senkronize edilmiş olanlardan daha iyi performans gösterir.
- Hashtable, boş anahtarlara veya değerlere izin vermez. HashMap, bir boş anahtara ve herhangi bir sayıda boş değere izin verir.
- Örneğin; HashMap'in alt sınıflarından biri LinkedHashMap'tir, bu nedenle tahmin edilebilir yineleme sırası istemeniz durumunda (varsayılan olarak ekleme sırasıdır), HashMap'i LinkedHashMap için kolayca değiştirebilirsiniz. Bu olamaz Hashtable kullanıyorsanız kolay.

Senkronizasyon benim için bir sorun değilse, HashMap kullanmayı tercih ederim. Bir sorun haline gelirse, tercih ederim Collections.synchronizedMap () veya ConcurrentHashMap.

- Hem Hashtable hem de HashMap, Harita arabirimini uygular ve her ikisi de Anahtar ve Değer'dir.
- HashMap iş parçacığı güvenli değildir, Hashtable iş parçacığı açısından güvenli bir koleksiyondur.
- HashMap senkronize olmadığı için ikinci önemli fark performanstır. Hashtable'dan daha iyi performans gösterdi. → Collections.synchronizedMap (... Harita ...);

**34. İstisna ile nasıl başa çıkarsınız?**

İstisnayı ele almak için dene-yakala yaklaşımını kullanırdım

• 1- Bir dene-yakala bloğunun içine bir istisna oluşturabilecek kodumu koyardım. Try-catch bloğu ile bir istisna veya kurtarma adımlarımı gerçekleştirmeyi deneyin. Ayrıca gerekirse multi veya Union Catch blokları kullanabilirim

• 2- throws anahtar sözcüğünü de kullanabilirim. ANCAK bu, benim yöntemimi arayan herkesin artık onu da halletmesi gerektiği anlamına geliyor!

• 3- Diğer bir yol da AutoCloseable'dır: Try bildiriminde AutoCloseable olan referansları yerleştirdiğimizde, kaynağı kendimiz kapatmamız gerekiyor. Yine de, istediğimiz başka türden bir temizlik yapmak için yine de bir nihayet bloğu kullanabiliriz. Deneyin-ile



## Sayfa 52

**35. TreeSet ve TreeMap**

• TreeSet: Yalnızca benzersiz değerler içerebilir - artan düzende sıralanır

• TreeMap: yalnızca benzersiz anahtarlar içerebilir. - anahtarlar artan düzende sıralanır

**36. final vs final vs nihayet?**

• **final** → bir anahtar sözcüktür ve sınıf, yöntem ve değişken üzerinde kısıtlamalar uygulamak için kullanılır.

o final Sınıfı Devralınamaz

o son Yöntem Geçersiz Kılınamaz

o son Değişken değeri DEĞİŞTİRİLEMEZ.

• **nihayet** → bir bloktur ve önemli kodu yerleştirmek için kullanılır, istisna ele alınsın veya işlenmesin çalıştırılır

• **finalize** → bir yöntemdir ve Object is Garbage toplanmadan önce temizleme işlemini gerçekleştirmek için kullanılır.

**37. Java'da Hata ve İstisna Arasındaki Fark?**

• Hem Hata hem de İstisna, Java'da Throwable'dan türetilmiştir.

• Hata, genellikle ele alınamayan hataları temsil eder.

Örnekler için: OutOfMemoryError, NoClassDefFoundError

• Öte yandan, İstisna, yakalanabilen ve dağıtılabilen hataları temsil eder.

Örnekler için> IOException, NullPointerException

• İstisna, işaretlenmiş ve işaretlenmemiş İstisna olmak üzere iki kategoriye ayrılır. Kontrol Edilmiş İstisna zorunlu gerektirir bunu işlemek için dene-yakala kod bloğu. Denetlenmemiş İstisna çoğunlukla programlama hatalarını temsil eder (NullPointerException veya Çalışma zamanı istisnası)

• Hatalar kontrol edilmeyen istisnalardır ve geliştiricinin bunlarla herhangi bir şey yapması gerekmez

• **Tüm Hatalar İstisnadır, ancak tersi doğru değildir.**

• Genel olarak Hatalar, hiç kimsenin ne zaman olduğunu kontrol edemediği veya tahmin edemediği durumlardır, diğer yandan İstisna tahmin edilebilir ve ele alınabilir

**38. Java'da RuntimeException ve CheckedException arasındaki fark nedir?**

• İstisna, Çalışma Zamanı (işaretlenmemiş) İstisna ve CheckedException olmak üzere iki kategoriye ayrılır.

• RuntimeException ve CheckedException arasındaki temel fark, işlemek için try-catch sağlamanın zorunlu olmasıdır. CheckedException durumunda RuntimeException zorunlu değildir.

• NullPointerExceptio, ArrayIndexOutOfBound, ClassNotFoundException gibi en yaygın İstisnalardan bazıları, IOException.

Öncelikle, Java İstisnalarının kontrolsüz olarak da bilinen RuntimeException iki kategoriye ayrıldığını hatırlatmak istiyorum.



## Sayfa 53

İstisna ve kontrol edilen (derleme zamanı) İstisna.

RuntimeException ile kontrol edilen İstisna arasındaki temel fark, try catch veya try catch sağlamak zorunludur.

kontrol edilen İstisnayı işlemek için blok ve bunu yapmamak derleme zamanı hatasıyla sonuçlanırken, RuntimeException olması durumunda bu zorunlu değildir.

NullPointerException, ArrayIndexOutOfBoundException gibi en yaygın İstisnalardan bazıları işaretli değildir ve bunlar

java.lang.RuntimeException'dan türetilmiştir.
Kontrol edilen İstisnalar için popüler bir örnek ClassNotFoundException ve IOException'dur ve yapmanız gereken budur.
Birçoğu IOException attığı için Java'da dosya işlemleri gerçekleştirirken bir try catch nihayet bloğu sağlayın.
Kişisel fikrimi sorarsam, Kontrol Edilmiş İstisnalar'ın, kazan plakası kodunu.
try-catch nihayet blok.

**39. Java'da fırlatma ve fırlatma arasındaki fark nedir?**

- throw ve throws, Java programlama dilinin Exception özelliğiyle ilgili iki anahtar sözcüktür.
- throw anahtar sözcüğü açıkça bir istisna atmak için kullanılır, diğer yandan, bir istisna bildirmek için throws anahtar sözcüğü kullanılır

  bu, dene-yakala bloğuna benzer şekilde çalıştığı anlamına gelir.
- Sözdizimini atmadan daha iyi görürsek, ardından bir Exception sınıfı atışı örneği gelir ve ardından istisna sınıfı adları gelir.
- yeni ArithmeticException ("Aritmetik İstisna") oluşturun; ArithmeticException oluşturur;
- throw anahtar sözcüğü yöntem gövdesi için kullanılırken, özel durumu bildirmek için yöntem imzasında atmalar kullanılır.

Her ikisi de Java'nın İstisna özelliği ile ilgili iki anahtar kelimedir. Atmak ve atmak arasındaki temel farkı hatırladığım kadarıyla
atar, kullanım ve işlevselliğindedir.

- throws, yöntem imzasında muhtemelen herhangi bir yöntem tarafından atılan İstisnayı bildirmek için kullanılır, örneğin

```
public void shutdown (), IOException {
        yeni IOException ("Kapatılamıyor");
 }
```

Ancak fırlatma, aslında Java kodunda İstisna atmak için kullanılır.

```
Yeni İstisna atın ("başlatılamıyor");
```

Diğer bir deyişle; throws anahtar sözcüğü herhangi bir yerde kullanılamaz istisna yöntemi imzası atma anahtar sözcüğü içinde kullanılabilir
yöntem veya statik başlatıcı bloğu, yeterli istisna işleme sağladı.
Oh, throw ile ilgili bir şey daha hatırlıyorum, throw anahtar sözcüğü break kullanmadan bir switch ifadesini kırmak için de kullanılabilir.
anahtar kelime

**40. Nesne ve Sınıf Arasındaki Fark?**

- Sınıf, istediğiniz kadar nesne oluşturabileceğiniz bir plan veya şablondur. Nesne, bir sınıfın üyesi veya örneğidir
- Sınıf, class anahtar sözcüğü kullanılarak bildirilir, Object esas olarak yeni anahtar sözcük aracılığıyla oluşturulur.

Sınıf, nesneler için bir şablondur. Bir sınıf, geçerli bir değer aralığı ve bir varsayılan değer dahil olmak üzere nesne özelliklerini tanımlar. Bir sınıf
ayrıca nesne davranışını açıklar. Bir nesne, bir sınıfın üyesi veya "örneği" dir ve tümünün içinde bulunduğu durumlara ve davranışlara sahiptir.
özellikler, sizin açıkça tanımladığınız veya varsayılan ayarlarla tanımlanan değerlere sahiptir.
Sınıf - Bir sınıf, türünün nesnesinin desteklediği davranışı / durumu tanımlayan bir şablon / plan olarak tanımlanabilir.
Bunları karşılaştırırsak pek çok farklılık vardır, ancak bunlardan bazılarının bilinmesi önemli olduğunu söyleyeyim;

- Java'da yeni anahtar kelime, newInstance () yöntemi, clone () yöntemi, fabrika gibi nesne oluşturmanın birçok yolu vardır.

  yöntem ve seriyi kaldırma. Java'da class anahtar sözcüğünü kullanarak sınıfı tanımlamanın tek bir yolu vardır.
- Nesne, ihtiyaca göre birçok kez oluşturulur. Sınıf bir kez ilan edilir.
- Nesne, bir sınıfın örneğidir. Sınıf, nesnelerin oluşturulduğu bir plan veya şablondur.
- Nesne fiziksel bir varlıktır. Sınıf, mantıksal bir varlıktır.



## Sayfa 54

**Örneğin** :

| | | | |
|---|---|---|---|
| **Sınıf** : İnsan | **Nesne** : Erkek, Kadın | **Sınıf** : Cep telefonu | **Nesne** : iPhone, Samsung, Moto |
| **Sınıf** : Meyve | **Nesne** : Elma, Muz, Mango, Guava | **Sınıf** : Yiyecek | **Nesne** : Pizza, Burger, Samosa |

**41. StringBuffer ve StringBuilder?**

- Temel fark, StringBuilder'ın senkronize edilmediği sırada StringBuffer'ın senkronize edilmesidir. Yani, StringBuilder olabilir

  eşzamanlı olarak aranır. Ve bu, StringBuilder'ı daha verimli hale getirir.
- StringBuffer senkronize edildi, StringBuilder senkronize değil
- StringBuilder, StringBuffer'dan daha verimlidir
- Yapıcı;
  - o StringBuilder () → **16** başlangıç **kapasitesine** sahip boş bir dize oluşturdu .
  - o StringBuilder (str str) → belirtilen dizeyi bir StringBuilder oluşturdu.
  - o StringBuilder (int length) → uzunluk olarak belirtilen kapasiteye sahip boş bir dizge oluşturdu.
- Yöntem;
  - o StringBuilder str = new StringBuilder ("Merhaba");
  - o str.append ("Java"); → // Merhaba Java
  - o str.insert (1, "Java"); → // HJavaello
  - o str.replace (1,3, "Java"); → // HJavalo
  - o str.delete (1,3); → // Merhaba
  - o str.reverse (); → // olleH

```
string str = "Merhaba";
ters dize = “”;
```

```
for (int i = str.length () - 1; i> = 0; i -) {
    ters + = str.charAt (i);
}
sysout (tersine çevrilmiş);
```

**42. Kesinleştirme () nedir?**

- finalize () yöntemi, java.lang.Object sınıfının korumalı ve statik olmayan bir yöntemidir.
- Bu yöntem, java'da oluşturduğumuz tüm nesnelerde mevcuttur.
- Bu yöntem, nesneden kaldırılmadan önce bir nesne üzerinde bazı son işlemleri veya temizleme işlemlerini gerçekleştirmek için kullanılır.

  hafıza.
- Bir nesne yok edilmeden önce gerçekleştirmek istediğimiz işlemleri korumak için finalize () yöntemini de geçersiz kılabiliriz.

  Çağrılabilir. object.finalize ();

**43. Son anahtar kelime nedir?**

- final anahtar sözcüğü Class ile birlikte başka hiçbir sınıfın onu genişletemeyeceğinden emin olmak için kullanılır, örneğin String sınıfı sondur ve yapamayız

  uzat.
- Çocuk sınıflarının onu geçersiz kılamayacağından emin olmak için final anahtar kelimesini yöntemlerle birlikte kullanabiliriz.
- final anahtar sözcüğü, yalnızca bir kez atanabildiğinden emin olmak için değişkenlerle birlikte kullanılabilir. Ancak, durumu

  değişken değiştirilebilir, örneğin, bir nesneye yalnızca bir kez son bir değişken atayabiliriz, ancak nesne değişkenleri

  daha sonra değiştirin.
- Java arayüz değişkenleri varsayılan olarak nihai ve statiktir.

**44. Statik anahtar kelime nedir?**

- static anahtar sözcüğü sınıf düzeyi değişkenlerle birlikte kullanılabilir ve genel hale getirilebilir, yani tüm nesneler aynı değişkeni paylaşacaktır.
- static anahtar sözcüğü yöntemlerle de kullanılabilir. Statik bir yöntem yalnızca sınıfın statik değişkenlerine erişebilir ve yalnızca

  sınıfın statik yöntemleri.



## Sayfa 55

**45. system.gc () nedir?**

- Belleği boşaltmak için Çöp toplayıcıyı çalıştırması için JVM'ye bir istek
- Her zaman işe yaramıyor

Java.lang.System.gc () yöntemi çöp toplayıcısını çalıştırır. Bunu aramak, Java Sanal Makinesi'nin
Şu anda kapladıkları hafızayı hızlı bir şekilde yeniden kullanıma hazır hale getirmek için kullanılmayan nesneleri geri dönüştürme çabası.
Bu bir komut değil, bir rica. Bu isteği yerine getirmek çöp toplayıcıya kalmıştır.

**46. Önemli String Metotları?**

**47. IS-A ve HAS-A ilişkisi arasındaki fark nedir?**

- **IS-A** , kalıtıma dayanır → Bu şey, o şeyin bir türüdür
- **HAS-A** ilişkileri kullanıma dayalıdır
  - o Örn: A Sınıfı HAS -AB, Sınıf A'daki kodun B sınıfının bir örneğine referansı varsa

```
public Horse {
    özel Halter myHalter;
    public void jump () {
                Sysout "atlıyorum"
```

• Horse sınıfından gelen atlama yöntemini kullanmak için bir Halter örnek değişkenini çağırıyorsunuz - bunun yaptığı şey şu ki

Atın HAS-A Halter olduğu anlamına gelir

• Horse sınıfının Halter'ı vardır çünkü Horse, Halter türünde bir örnek değişkeni bildirir. Kod, üzerinde tie () işlevini çağırdığında

Horse nesnesinin Halter örnek değişkeni -}

• Soyut sınıfın yapıcıları varken arayüzün yapıcıları yoktur



## Sayfa 56

**48. Yineleyici nedir ve her döngü için arasındaki fark nedir?**

• Yineleyici, dizi ile değil ArrayList ile çalışır.

• Öğeleri yinelememize yardımcı olacaktır.

• Fark, yineleyici ile yineleme sırasında listede değişiklikler (öğeyi kaldır) yapabilmenizdir.

• her döngü içinde listemizde değişiklik yapamayız

**49. Java Collection Framework**

İki tür Koleksiyon (Bunları karıştırmamaya dikkat edin)

✪ **java.util.Collection** - Set ve List genişlemesinden arayüz (uygulama değil)

✪ **Set** *(Eşsiz şeyler)* - ÇİFTLERE İZİN VERMEZ. Seti Uygulayan Sınıflar;

♦ **HashSet** → Yinelemeleri istemediğinizde ve yinelediğinizde siparişi önemsemediğinizde kullanın

vasıtasıyla

o Sırasız ve Sıralanmamış

♦ **LinkedHashSet** → **HashSet'in sıralı** sürümü ve yineleme sırasını önemsediğinizde HashSet üzerinden kullanın

♦ **SortedSet**

♦ **TreeSet** → Öğeler, öğelerin doğal sırasına göre artan sırada olacaktır.

o Doğal düzenin kendi kurallarını uygulamak için kurucuyu da özelleştirebilir

✪ **Liste** *(şeylerin listesi)* - dizini önemsiyor. List'i uygulayan sınıflar;

♦ Bağlantılı **Liste** → Dizin konumuna göre **sıralanır** ve öğeler birbirine çift bağlıdır

o Yığın ve kuyruğu uygulamak için iyi bir seçimdir

o ArrayList'ten daha yavaş yineler, ancak hızlı ekleme ve silme

♦ **Vektör** → ArrayList ile aynı AMA vektör yöntemleri senkronize edilir (iş parçacığı güvenli)

♦ **Dizi Listesi** → Hızlı yineleme ve Hızlı rastgele erişim ve sıralı (dizine göre)

o Ayrıca sıralanmamış (ancak sıralamak için Collections.sort () işlevini çağırabilir)

✪ **java.util.Collections** - koleksiyonlarla kullanılmak üzere statik yardımcı program yöntemlerini tutan bir sınıf; Ekle, kaldır, içerir, içerir

boyut ve yineleyici vb.

• **Harita** *(benzersiz kimliği olan şeyler)* → Önemli: Harita ile ilgili sınıfların ve arayüzlerin hiçbiri Koleksiyon formunu genişletmez.

Haritanın uygulama sınıfları Koleksiyon değil, "koleksiyonlar" olarak düşünülür. Map'i uygulayan sınıflar;

♦ **Hashtable**

o HashMap ile aynı ANCAK HashTable yöntemleri senkronize edilir (UNUTMAYIN. YALNIZCA YÖNTEMLER

SENKRONİZE EDİLMİŞ, SINIFLAR VEYA DEĞİŞKENLER DEĞİL)

o Hashtable hiçbir şeyi BOŞ (BOŞ) almanıza izin vermez

♦ **LinkedHashMap**

o Kampanya siparişini (veya isteğe bağlı olarak erişim sırasını) korur

o Öğe eklemek / kaldırmak için HashMap'ten daha yavaş, ancak DAHA HIZLI İTERASYON

♦ **HashMap** → Sıralanmamış ve Sıralanmamış & Bir koleksiyonda bir boş ANAHTAR ve birden çok boş değere izin verir

o KeySet ()

o Map.keySet () - bir dizi Anahtar döndürür

o Map.keySet (). size - anahtar sayısını döndürür

♦ **SortedMap** → TreeMap

• Set, List ve Map uygulama sınıfları ASLA hem sıralanamaz hem de sıralanamaz, tüm diğer kombinasyonlar olabilir.

**50. float'ı String'e nasıl dönüştürebilirim?**

```
float f = Float.parseFloat ("25");
Dize s = Float.toString (25.0f);
```



## Sayfa 57

**51. Diyelim ki "int b = 3; ve int a = 4;" onları nasıl değiştirebilirsin?**

```
// tek satırlık yöntemler
a = a ^ b ^ (b = a);
b = (a + b) - (a = b);
a + = b - (b = a);

int temp = a; // geçici değişken
a = b; b = sıcaklık;
```

**52. Yazmayı biliyor musunuz? Oyuncu seçimi nedir?**

• **Otomatik kutulama** → ilkel bir değer alıp sarmalayıcı sınıf nesnesine int i = 10 atadığınız bir süreçtir;

```
Tamsayı n = i;
Tamsayı num = 200;
Tamsayı num2 = new Integer (400); // BOXING YOK
```

• **Un-boxing** → Wrapper sınıf nesnesini alıp ilkele dönüştürdüğünüz bir süreçtir.

```
Tamsayı num2 = yeni Tamsayı (400);
Tamsayı num = 200;
int i = num2;
```

• Bir türden bir değerin başka türden bir değişkene atanması, Type Casting olarak bilinir.

**53. Bu programın çıktısı nedir?**

```
for (int i = 0; i <3; i ++) {
    for (int j = 3; j> = 0; j--) {
                eğer (i == j)
                devam et;
                System.out.println (i + "" + j);
    }
}
```

Çıkış: 1 0 2 3 2 1 2 0

**54. Projenizde soyut bir sınıfı nasıl kullanıyorsunuz bana bir örnek verin?**

• Bu kavramlar genellikle çerçeve geliştirmede kullanılır. Soyut sınıf, ortak bir süper sınıfı tanımlamada kullanılır

çerçevenin Sayfa Nesne Modeli katmanını yazarken. Genellikle hepsine sahip olmak için BasePage adında soyut bir sınıf oluştururuz.
Bu sınıf örneğinde yazılan her sayfa için ortak üyeler **getPageTitle () .**

• Daha sonra her Sayfa sınıfı (HomePage, LoginPage, DashboardPage vb.) BasePage'den devralır. Bazen ihtiyaç duyulabilir

üst sınıfta uygulanan yöntemlerin davranışını değiştirir. Dolayısıyla, alt sınıf, bu yöntemi geçersiz kılma özgürlüğüne sahiptir.
polimorfizm kullanın. Abstract sınıfını gerçek projelerde böyle kullanıyoruz.

**55. Değere göre geçirme ile referansla geçirme arasındaki fark nedir? değere göre geçmek ve referans ile geçmek?**

• Değere göre geçiş, işlev parametresinin değerinin belleğinizin başka bir konumuna kopyalandığı anlamına gelir ve

İşlevinizdeki değişkene erişirken veya değiştirirken, yalnızca kopyaya erişilir / değiştirilir ve orijinal değer
dokunulmadan bırakılır. Değere göre geçiş, değerlerin çoğu zaman nasıl aktarıldığıdır.

• Başvuru yoluyla geçiş, değişkenin bellek adresinin (bellek konumuna bir işaretçi),

işlevi. Bu, bir değişkenin değerinin aktarıldığı değere göre geçişten farklıdır. Örneklerde hafıza adresi
/ myAge 106'dır. myAge işlevini artırırken, işlev içinde kullanılan değişken olan artışAgeByRef (yaş
bu örnek) hala orijinal myAge değişkeni ile aynı bellek adresine işaret etmektedir (İpucu:
fonksiyon parametresi, bir değişkenin referansını / işaretçisini elde etmek için birçok programlama dilinde kullanılır).



**Sayfa 58**

# SELENYUM

**1. Selenyum nedir ve nelerden oluşur?**

• Selenium, otomatik web testi için bir araç paketidir. Tarafından bestelendi;

o Selenium IDE (Entegre Geliştirme Ortamı); kayıt ve oynatma için çalışan bir Firefox eklentisi.
o Selenium RC (Uzaktan Kumanda) (1.0); bir test aracıdır ve web uygulamasını otomatikleştirmek için JS üzerinde çalışmak için kullanılır. (2004)
o WebDriver (2.0); bir web otomasyon çerçevesidir ve testlerinizi farklı tarayıcılarda yürütmenize olanak tanır. (2011)
o Selenyum Izgara; testlerin birden çok makinede paralel olarak yürütülmesine izin verir.

**2. Selenium'un avantajları nelerdir?**

• Selenium açık kaynak kodludur ve herhangi bir lisans maliyeti olmaksızın kullanımı ücretsizdir
• Java, Ruby, Python, C # gibi birden çok dili destekler ...
• Çoklu tarayıcı testini destekler
• İyi miktarda kaynağı vardır ve topluma yardım eder
• Windows, Mac, Linux gibi birçok işletim sistemini destekler ...
• Web uygulamasıyla etkileşim kurun