K147 KODLUYORUZ JAVA BOOTCAMP

Sorular

- **Java'nın platform bağımsızlığını nasıl sağladığını anlatınız. (5 PUAN)**
- **Java'da heap ve stack kavramlarını örneklerle açıklayın. (5 PUAN)**
- **String class'ı nasıl immutable olmayı sağlamaktadır örnek ve çizimlerle açıklayınız. (5 PUAN)**
- **Java neden çoklu kalıtımı desteklemez açıklayın? (5 PUAN)**
- **Build Tool nedir? Java ekosistemindeki build toolar neler açıklayın? (5 PUAN)**

Java'nın platform bağımsızlığını nasıl sağladığını anlatınız. (5 PUAN)

C, C# gibi diler derlendiğinde tüm makinalarda çalışabilmesi için 0 ve 1'lere çevrilir. Bu işlem işletim sisteminde olur ve biz 0 ve 1'lerden oluşan sisteme makine dili deriz. Kod eğer Windows'ta yazılmışsa aynı kod farklı makinelerde derlendiğinde örneğin Solaris'te derlenmişse farklı ya da Linux'te derlenmişse farklı oluyor. Bu farklılık yazılan kodun başka platformda çalışmaması anlamına geliyor. Yani kod Windows'ta yazıldığı için yalnızca Windows'ta derlenebiliyor. Burada asıl sorun bu dillerin işletim sisteminde derlenmesidir. Bu sebeple platforma bağımlıdırlar.

JRE → Java'nın platformdan bağımsız olarak çalışabilmesi için JRE gereklidir. JRE içerisinde JVM ve Java kütüphanelerini barındırır. Java JVM (Java Virtual Machine) üzerinden çalışır. Java sanal makinesinin çalışma mantığını incelersek;

- Java kodları geliştirme ortamı tarafından yazım hatalarına karşı kontrol edilir. Hatalar giderilmişse sonraki aşamaya geçilir.
- JDK paketindeki derleyici (compiler) aracılığı ile java kodları bytecode denilen bir ara dilin kodlarına dönüştürülür.
- Üzerinde çalışılan sistem sistemdeki JVM bu bytecode'u yorumlar ve çalıştırır.

Her çalıştırıldığında kaynak kodun yeniden yorumlanması gibi nedenlerden dolayı nispeten yavaş olması dezavantajdır. Ama Java'nın platformdan bağımsız olması büyük avantajdır. "Bir kere yaz, her yerde çalıştır." felsefesine sahiptir.

JVM → Tüm platformlarda Java kodlarını çalıştırmak üzere geliştirilmiş ve hemen her platforma uygun sürümü olan bir bileşendir. Windows, Linux vb. platformlar için sürümleri mevcuttur.

JDK → programcıya yazılım geliştirme sürecinde gerekli olacak bileşenleri içeren bir pakettir.

Bytecode → Bytecode, makina diline benzetilen bir ara dildir.

Özetle Java JRE'de bulunan JVM ve Java kütüphaneleri sayesinde yazılan kodu kendi bytecode'una çevirerek platdormdan bağımsız olarak çalışabilir.

Java'da heap ve stack kavramlarını örneklerle açıklayın. (5 PUAN)

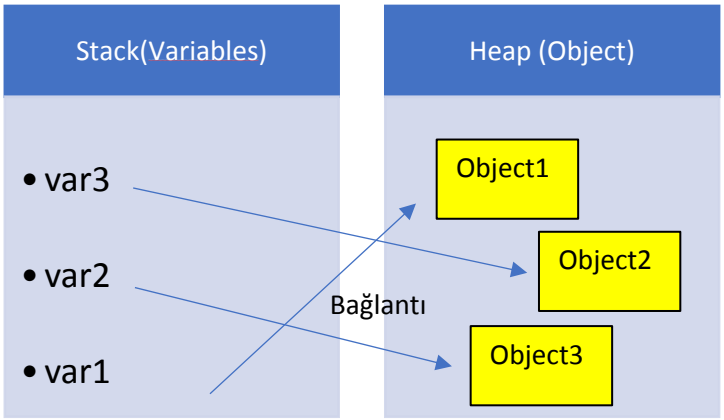
Stack ve Heap bellekte (ram) bulunan mantıksal yapılardır.

Heap → Herhangi bir Java sınıfından new operatörü ile bir nesne oluşturulduğunda, bu nesnenin bilgisayarın hafızasında konuşlandırıldığı alanın adıdır. Bu alanda oluşturulan nesnelere uygulamanın her yerinden ulaşılabilir. Referansı olmayan nesneleri kaldırarak bellek temizleme etkinliğini gerçekleştirmek için GarBage (çöp toplama) tarafından yönetilir.

- Heap ortak olarak kullanılır ve uygulama başlatıldığında başlar.
- Burada veriler karışık bir şekilde saklanır.
- Stack ile kıyaslandığında veriye erişmek maliyetlidir.
- Heap'de referans değerleri saklanır.

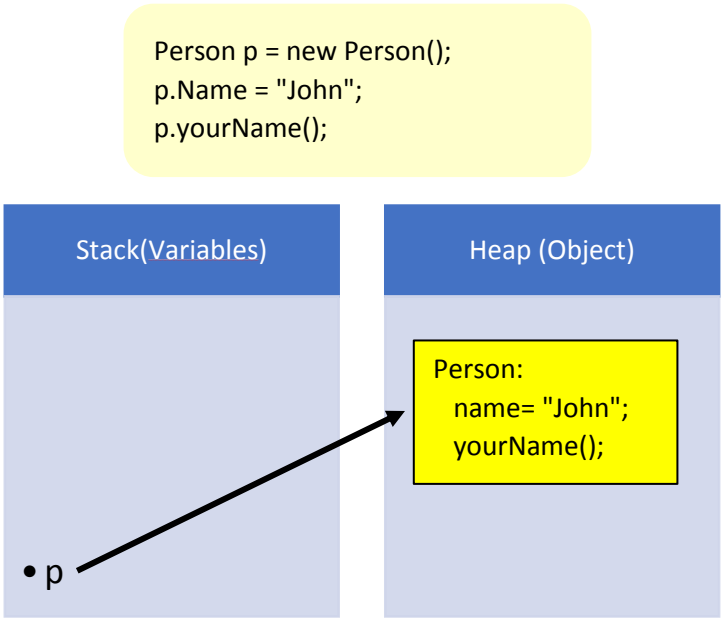
Stack → İş parçacıklarının yürütülmesi için kullanılır. Heap belleği yöntemlere, yerel değişkenlere ve referans değişkenlere ayrılmıştır. Belleğe LIFO (son giren ilk çıkar) sırasına göre başvurulur ve bu değerlere hızlı erişim sağlar. Bir yöntem çağırıldığında, yerel değişkenleri tutmak ve yöntemdeki diğer nesnelere referans vermek için bir blok oluşturur. Yöntem yürütmesini sonlandırdığında, oluşturulan blok kullanılmaz hale gelir ve bir sonraki yöntem için kullanılabilir olur.

- Stack'de değer tipleri (int, short, byte, long, decimal, double, float), pointer ve adresler saklanır.
- Stack daha hızlıdır. Ulaşılmak istenen veriler ard arda sıralanmış olur. Sırası gelmeden aradaki bir değer ile işlem yapamaz.
- Veriler artan yada azalan adres mantığında sıralanır.
- Veriler LIFO mantığında dizilir. Yani son gelen ilk olarak çıkar.
- Heap'te ki verilerler Garbage(çöp toplayıcı) algoritmasına bağlı iken stack'teki veriler hemen silinir.
- Class type (Sınıf tipi) değişkenler referans tiplerdir referans ettikleri model (referans) stack de değerleri ise heap de saklanır.



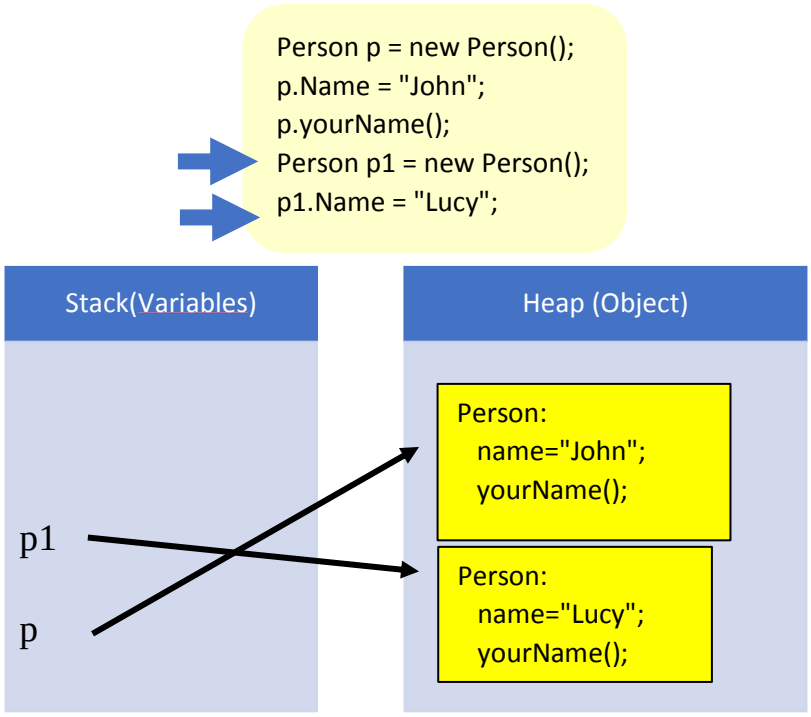
Şekil 1:Stack ve Heap

Stack tüm primitive değişkenlerin gittiği yerdir. Heap ise nesnelerin yaratıldığı yerdir. Heap'e giren bir nesneye erişmek için gereken bir değişken ve bağlantı oluşturulur. Bir nesneye erişmemiz gerekirse doğrudan erişemiyoruz bu yüzden var1'in işaretçi depoladığını düşünürsek, nesnenin nerede depolandığını nasıl anlayacağını bilir ve daha fazla nesne yaratırken bunlara atıfta bulunmak için daha fazla değişkene ihtiyaç vardır. Belirli bir değişkeni hangi nesneye erişmek istediğimizi görmek için kullanabiliriz.



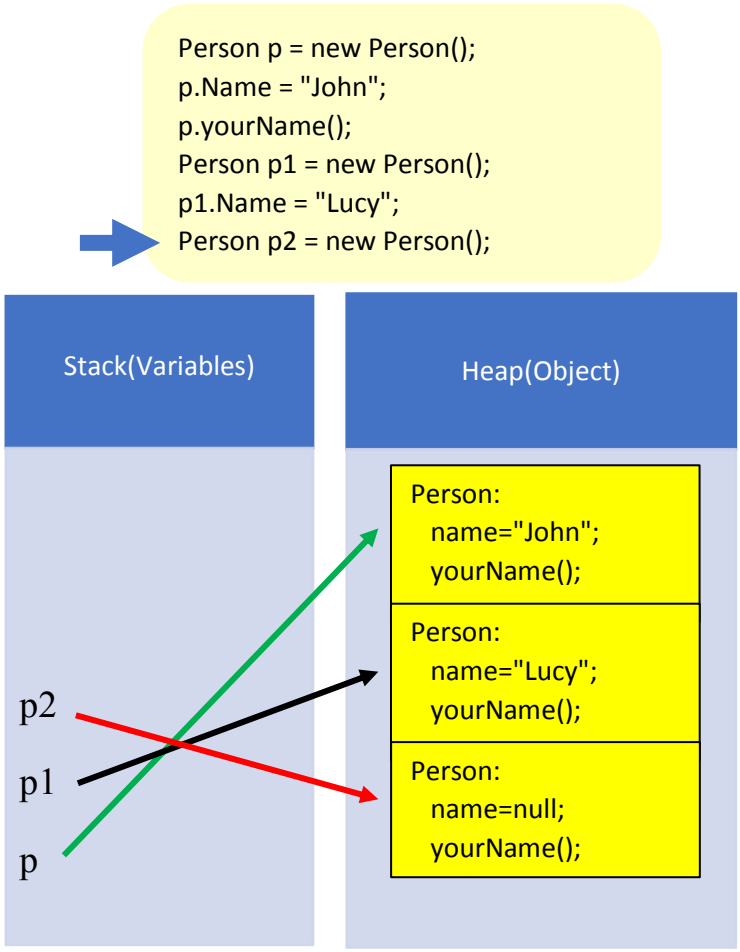
Şekil 2: Örnek 1

New anahtar sözcüğü yeni bir nesne oluşturmak için kullanılır. Bu komut kullanıldığında yeni bir nesne yaratılır. Nesne heap'te yaratılır ve p değişkeni stack'te yaratılır. p değişkeni stack'te bulunan nesneyi işaret eder. Bu sebeple p'yi Person nesnesine erişmek için kullanabiliriz. Nesneye ulaştığımızda nesneye ait özelliklere ve davranışlara da ulaşabiliriz. p.Name= "John"; ile name değişkenine değer atamış oluruz.



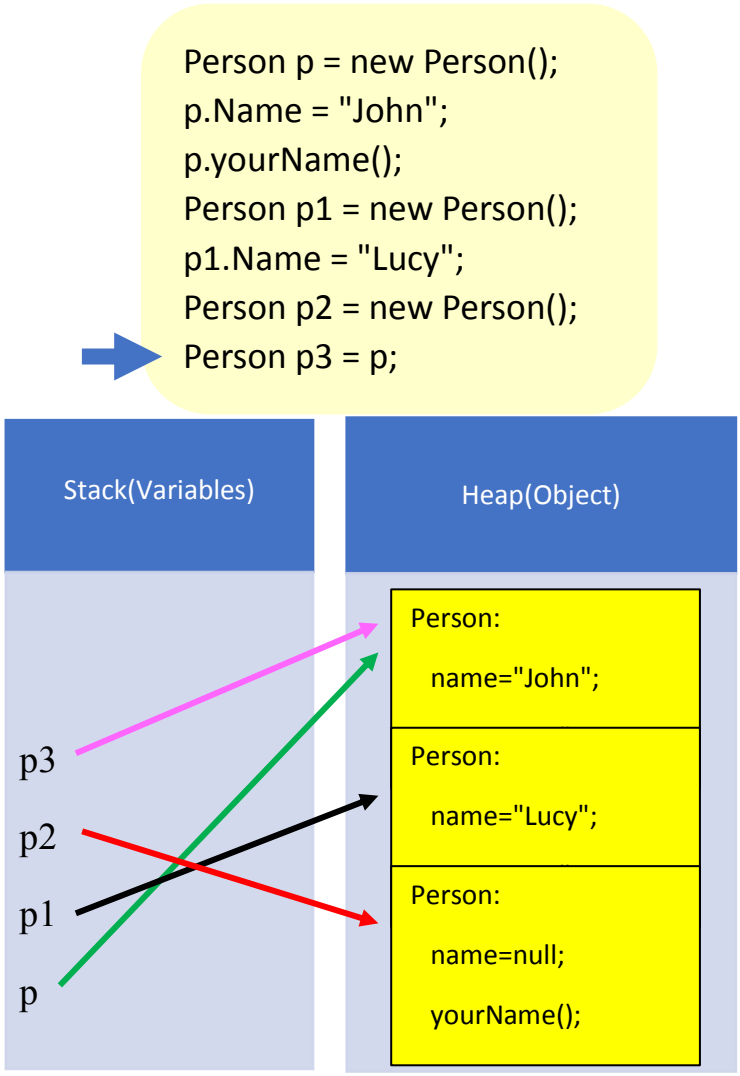
Şekil 3: Örnek 2

Yeni bir p1 nesnesi oluşturduğumuzda ve onun name değişkenine değer atadığımızda stack'de p1 adında yeni bir değişken oluşturulur. Heap'te ise name değeri Lucy olan yeni bir nesne oluşturulur. p1 ise yeni oluşturulan bu objeyi işaret edecektir.



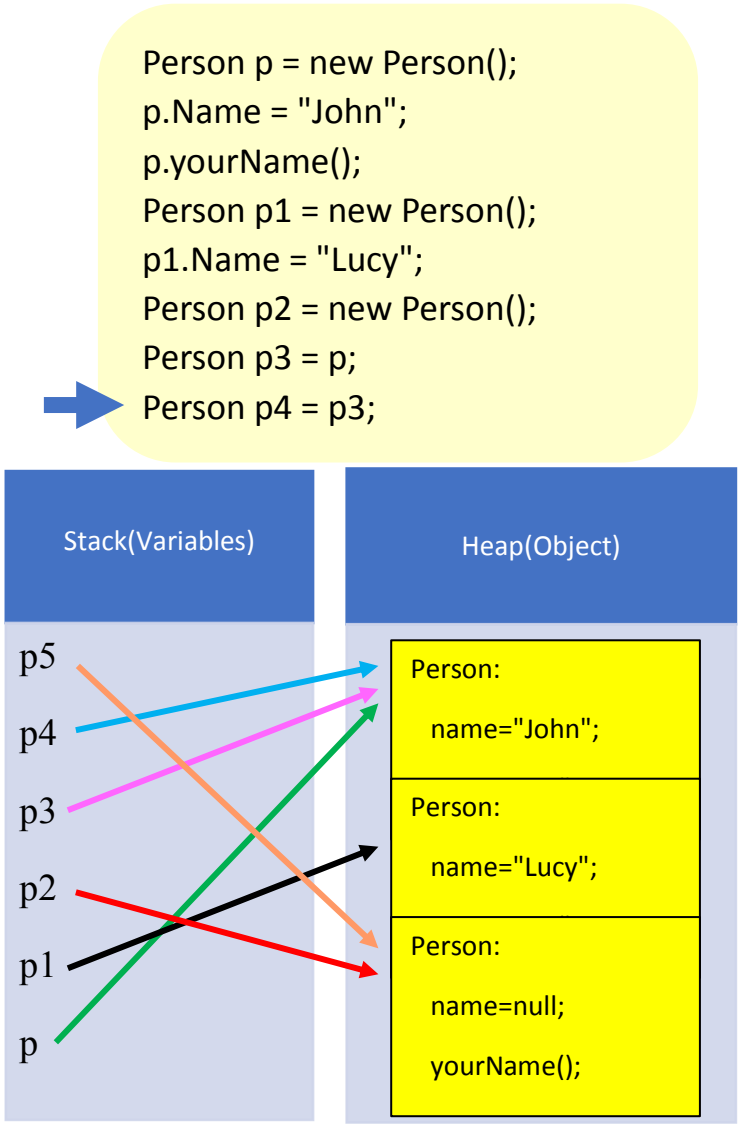
Şekil 4: örnek 3

Yeni bir p2 nesnesi oluşturduğumuzda stack'de p2 adında yeni bir değişken oluşturulur. Heap'te ise oluşturulan nesnede name değeri herhangi bir atama olmadığından dolayı null olur. p2 ise yeni oluşturulan bu objeyi işaret edecektir.



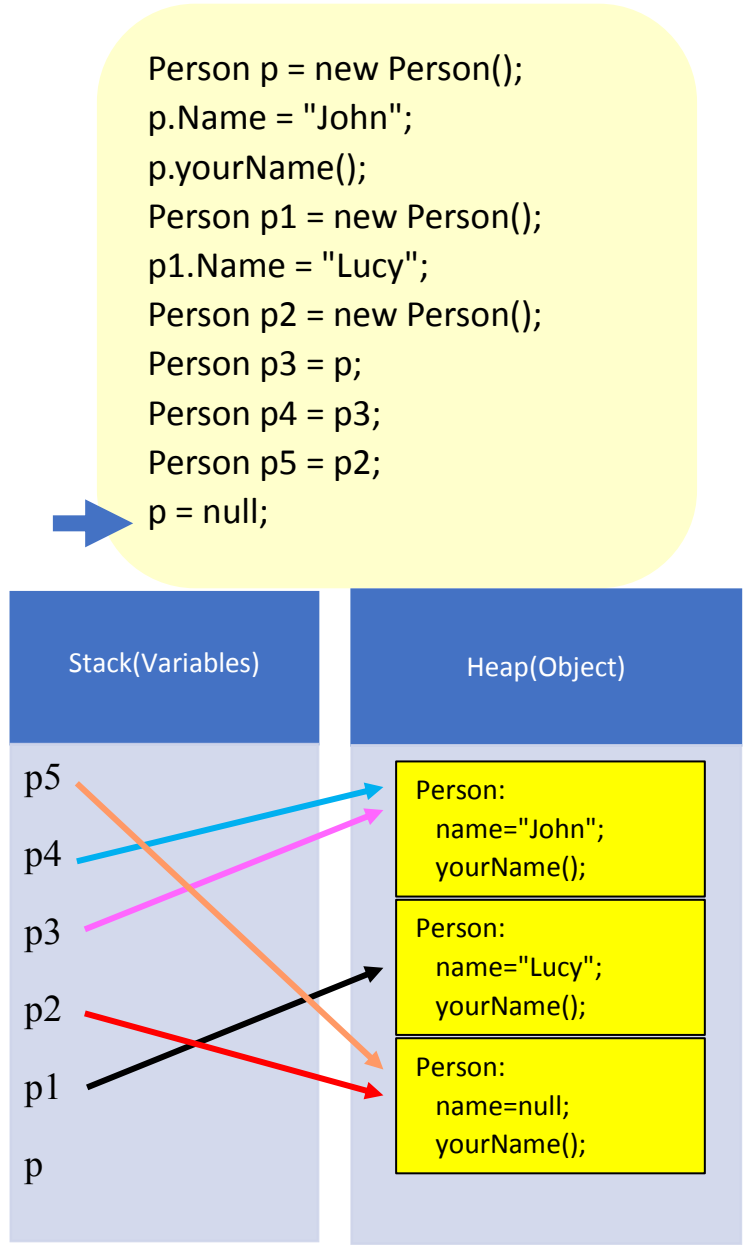
Şekil 5 :Örnek 4

p3 adında yeni bir değişken oluşturduğumuzda ve bu değişkene p değişkenini atadığımızda p3 değişkeni p değişkeninin işaret ettiği objeyi işaret edecektir.



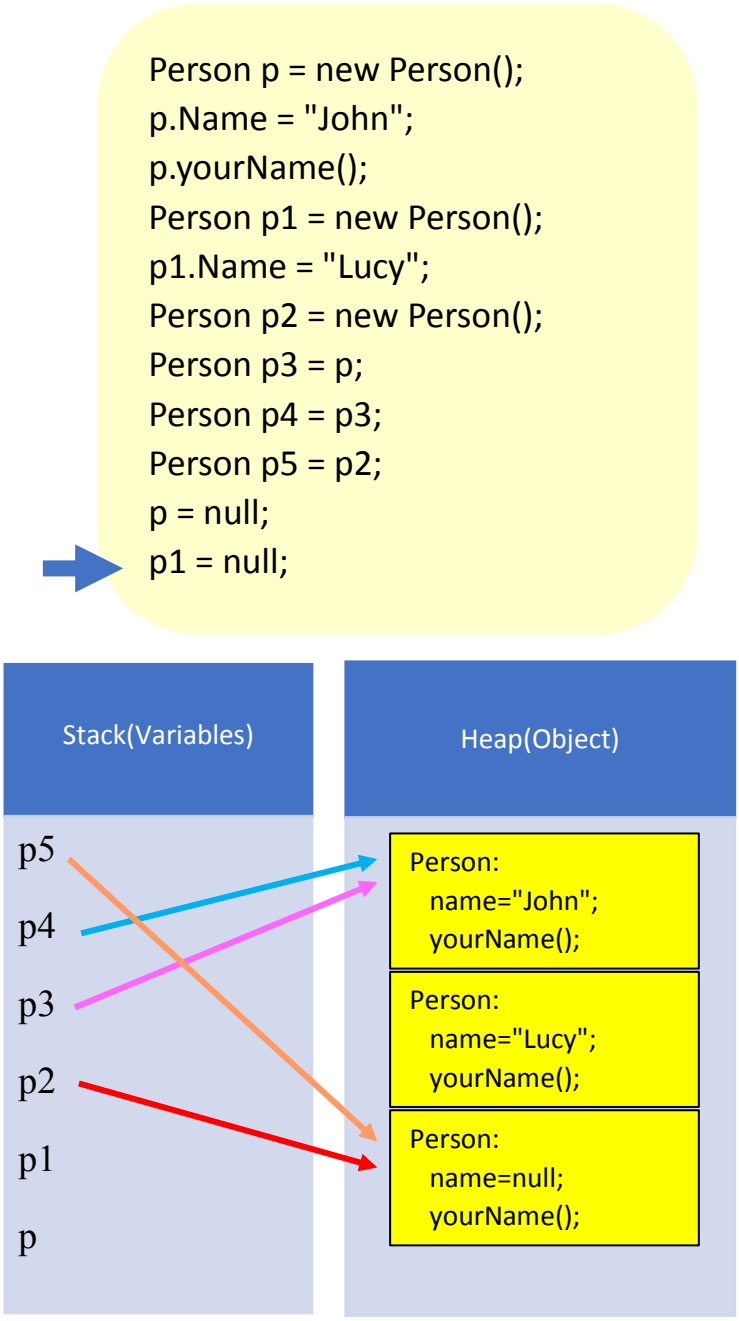
Şekil 6: Örnek 5

p5 adında yeni bir değişken oluşturduğumuzda ve bu değişkene p2 değişkenini atadığımızda p5 değişkeni p2 değişkeninin işaret ettiği objeyi işaret edecektir. Öyleyse buradan stack'de değişkenlerin obje adreslerini tuttuğunu söyleyebiliriz.



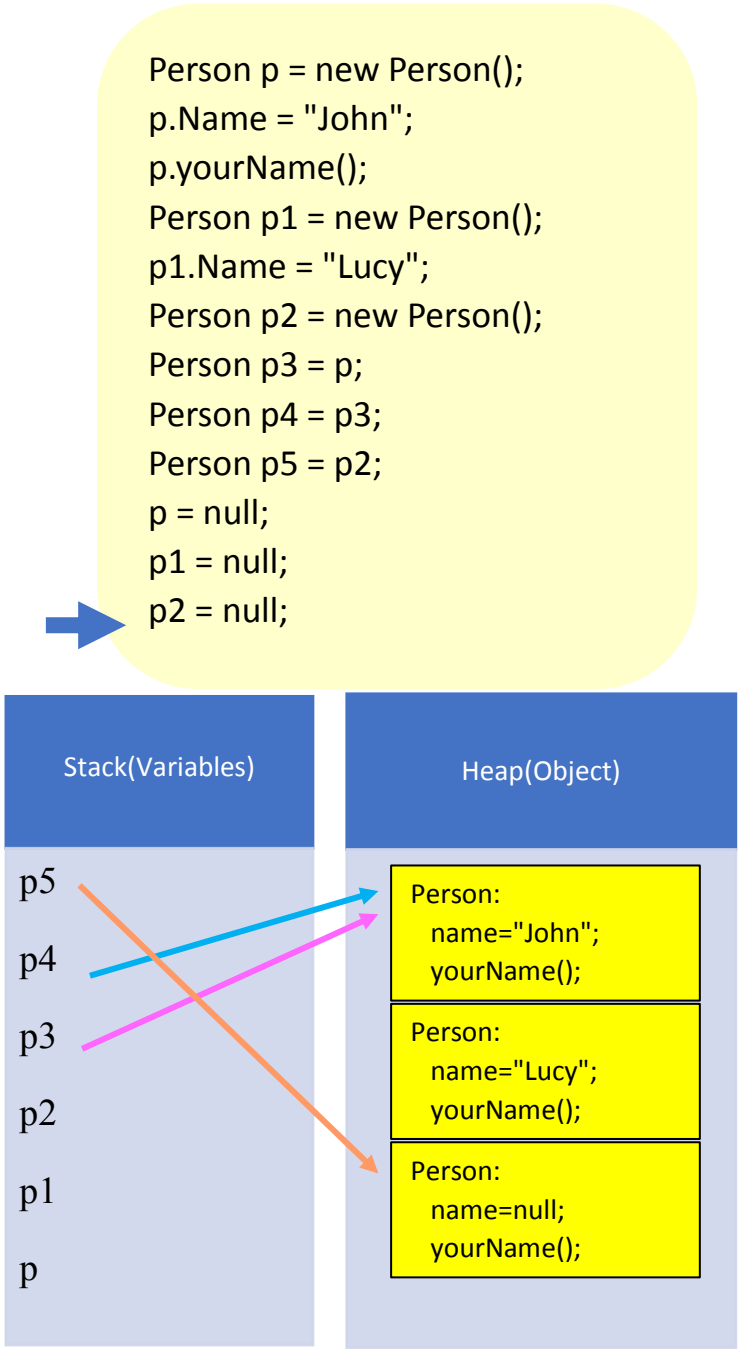
Şekil 7: Örnek 6

p=null ifadesi değişkenden bağlantıyı koparmaktır. Artık p üzerinden o nesneye erişemeyiz. Yani bu durumda p is equal to null yapıyoruz. Bu nedenle p'den Jhon'a olan bağlantı da kopmuş oluyor.



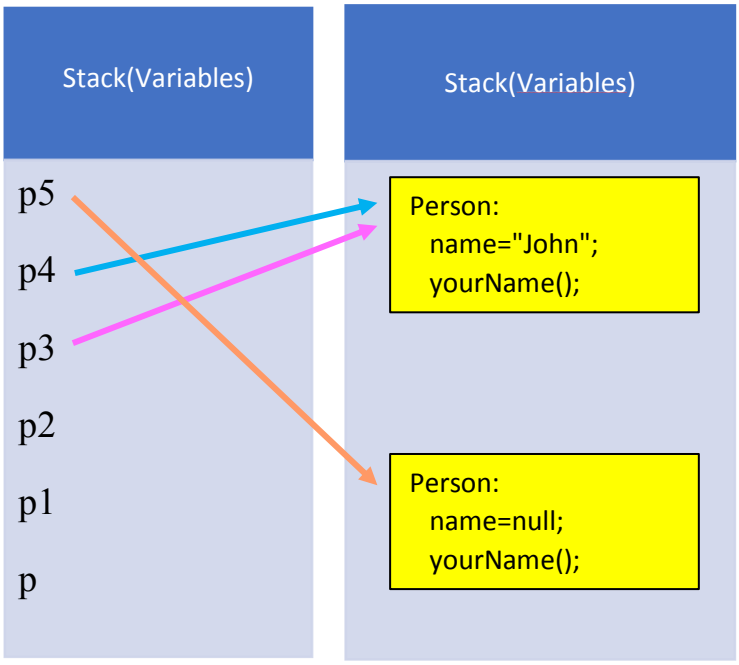
Şekil 8: Örnek 7

p1=null ifadesi değişkenden bağlantıyı koparmaktır. Artık p1 üzerinden o nesneye erişemeyiz. Yani bu durumda p1 is equal to null yapıyoruz. Bu nedenle p'den Lucy'e olan bağlantı da kopmuş oluyor.



Şekil 9: Örnek 8

p2=null ifadesi değişkenden bağlantıyı koparmaktır. Artık p2 üzerinden o nesneye erişemeyiz. Yani bu durumda p2 is equal to null yapıyoruz. Bu durumda Jhon kişisine p3 veya p4'den başvurulabilir. Her ikisi de tamamen aynı kişiyi işaret eder. P3 ve p4 aynı kişiyi ifade ediyor varsayalım ki adı değiştirmek için p3'ü kullandık ve Zeynep yaptık. Bu durumda p4'ü kullanarak isme ulaşmaya çalıştığımızda ismin Zeynep olduğunu görürüz. İsme ulaşma kısmına Java'da referansla çağırma deriz. Lucy kişisinin herhangi bir bağlantısı yok yani eğer bir nesnenin stack'den herhangi bir bağlantısı yok ise bu programdan o nesneye ulaşamayacağımız anlamına gelir. Bu nedenle bellekten tasarruf etmek için Java'da bir arka plan işlemi vardır.



Şekil 10: Örnek 9

Java'da ki arka plan işlemi Heap'i düzenli aralıklar ile tarar tüm nesneleri kontrol eder. Bir nesnenin Stack'ten bir referansı yok ise nesneyi görür ve işaretlenir. Böylece nesneyi fiilen silebilir. Buna Garbage denir. Bu işlemi ne zaman yapar bilemeyiz. Nesne silindikten sonra Heap'te ki bellekte alan açılmış olur.

String class'ı nasıl immutable olmayı sağlamaktadır örnek ve çizimlerle açıklayınız. (5 PUAN)

Javada **immutable** yani değişmez yapılar vardır. String, Boolean, Integer gibi bütün wrapper classlar immutable'dır. Immutable classlar bir kere oluşturulur ve değişmezler. Bu durumun bize sağladığı avantajlar ise şu şekilde belirtilebilir:

- Basit tasarım
- Güvenlik
- Verimlilik

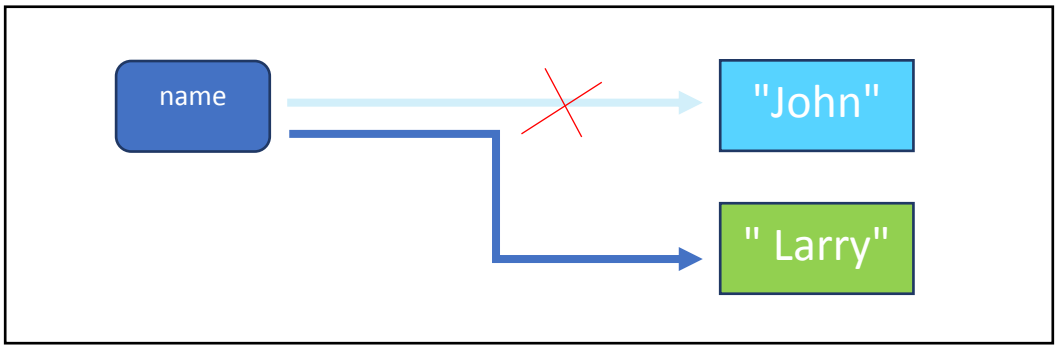
String class'ın immutable olmayı nasıl sağladığını incelersek;

```
public class ImmutableStrings{

    public static void main(String[] args){

        String name = "John";

    }

}
```

Şekil 1: String değişkeni oluşturma ve değer atama

name String nesnesinin kendisi değildir. " John " değeriyle oluşturduğu bellekteki String nesnesine bir referanstır.

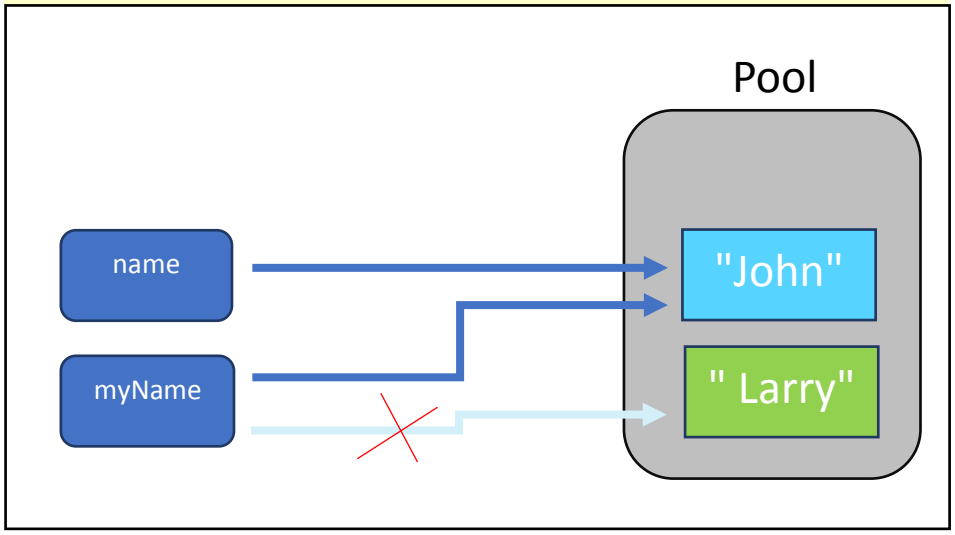
```
public class ImmutableStrings{

    public static void main(String[] args){

        String name = "John";

         name= "Larry";

    }

}
```



Şekil 2: String değişkenine yeni değer atama

Name değişkenine yeni bir değer atadığımızda Java bellekteki dize nesnesini değiştirmez. Bellekte " Larry " değeriyle yeni bir dize nesnesi yaratır. name değişkenini o dize nesnesine işaret edecek şekilde değiştirir. String nesnesi değişmezdir. String değişkenleri, istediğimiz String'i işaret edecek şekilde değiştirilebilir.

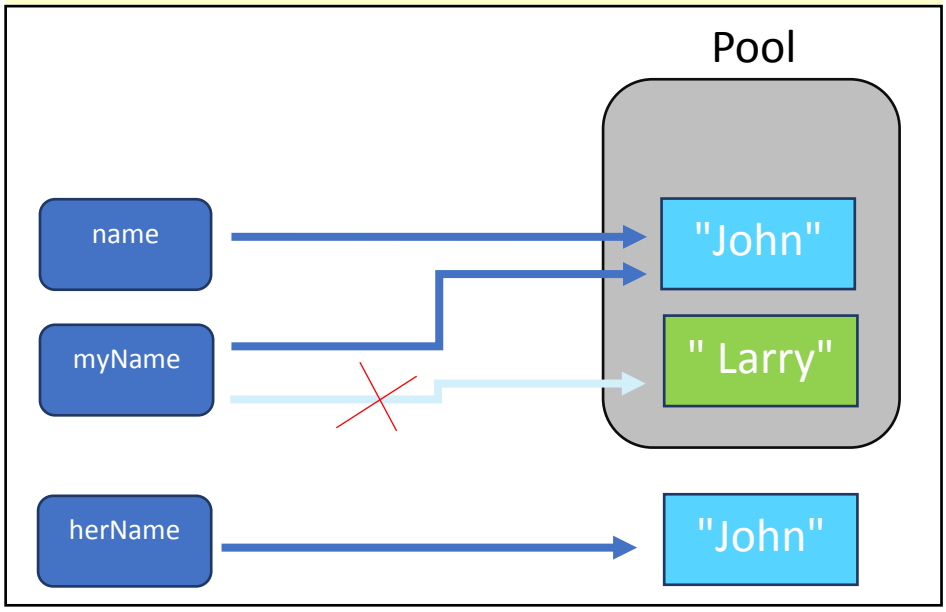
```
public class ImmutableStrings{

    public static void main(String[] args){

        String name = "John";

         String myName = "Larry";

         myName = "John";

    }

}
```



Şekil 3: İki String değişkenine aynı değer atanması

Java bir kez oluşturduğu "John" nesnesini ikinci kez oluşturmaz. Önce havuza(pool) bakar varsa String'i işaret edecek şekilde değiştirir. Yeni bir dize nesnesi oluşturmak için kullanacağı belleğin yarısını kullanmış olur. Bunu dizelerin değişmez olması sayesinde yapabiliyor ve bellekte tasarruf ediyor. Eğer değişebilir olsaydı yeniden "John " nesnesi üretmesi gerekirdi.

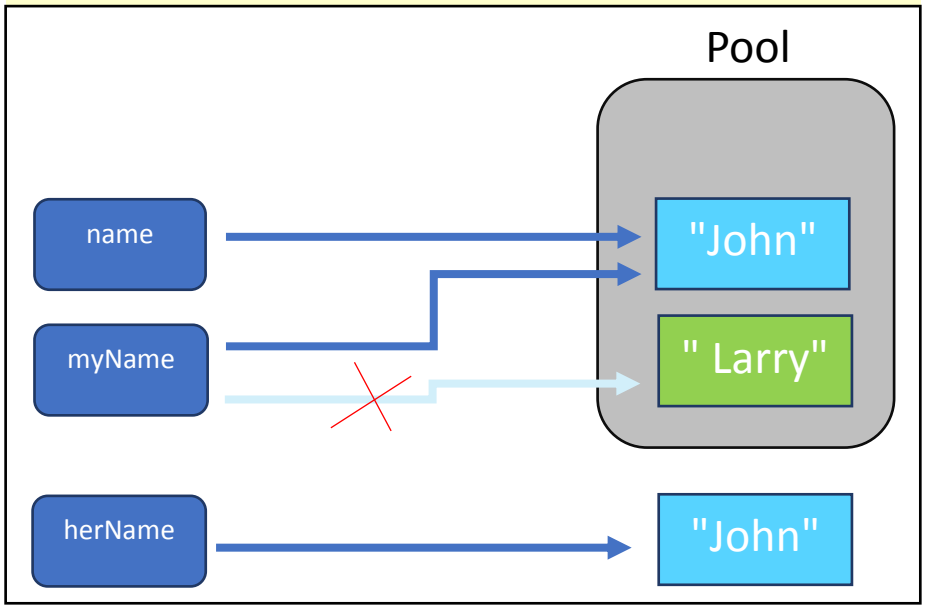
```
public class ImmutableStrings{
    public static void main(String[] args){
        String name = "John";
        String myName = "Larry";
        myName = "John";
        String herName = new String("John");
    }
}
```



Şekil 4: "new" anahtar kelimesini kullanma durumu

Java new anahtar kelimesini kullandığımız için nesneyi dize havuzundan kullanmayacaktır. Bunun yerine dize havuzunun dışında John değeriyle yeni bir nesne oluşturacaktır. Havuzda tamamen aynı değere sahip nesne bulunmasına rağmen Java new anahtar kelimesinden dolayı tamamen ayrı bir dize havuzu yaratacaktır.

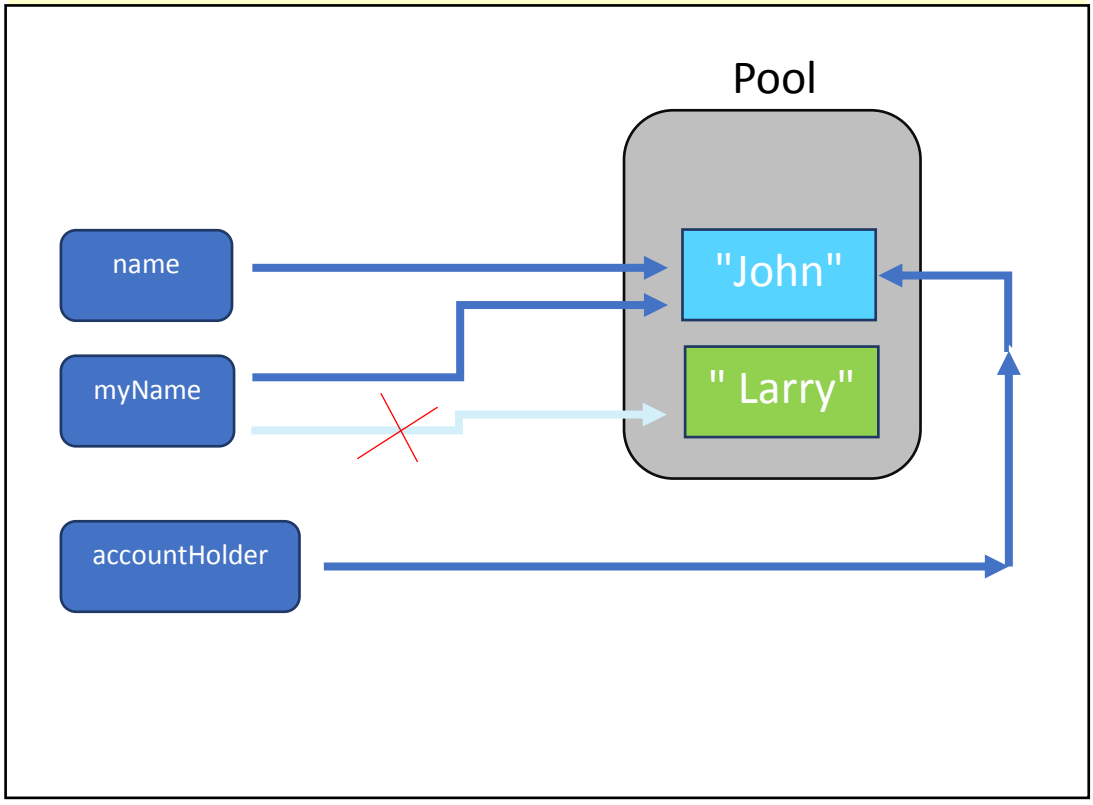
```
public class ImmutableStrings{
    public static void main(String[] args){
        String name = "John";
         String myName = "Larry";
         myName = "John";
         String herName = new String("John");
         System.out.println(name == myName);
         System.out.println(name == herName);
    }
}
```



Şekil 5: Kontrol çıktısı

```
//Aynı havuzu kullandıkları için bize "true" değerini verecektir.
System.out.println(name == myName);
//Farklı havuzu kullandıkları için bize "false" değerini verecektir.
System.out.println(name == herName);
```

```
public class ImmutableStrings{
    public static void main(String[] args){
        String name = "John";
         String myName = "Larry";
         myName = "John";
         String herName = new String("John");
         System.out.println(name == myName);
         System.out.println(name == herName);
         addMoneyToAccount(name, 5000);
}
    public static void addMoneyToAccount(String accountHolder, int moneyToAdd){
        //kodlar
    }
}
```



Şekil 6: Method'larda String'in değişmezliği

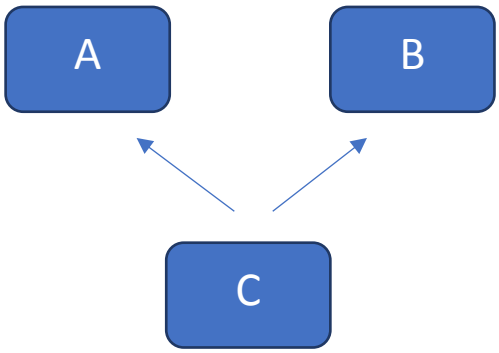
addMoneyToAccount(String accountHolder, int moneyToAdd) methodu çağırıldığında accountHolder değişkeni, bellekteki aynı dize nesnesine işaret edecektir.

Bir hesaba para ekleme işlemi yapıldığında. accountHolder değişkeninin işaret ettiği dize nesnesinin, dizelerin değişmez olmasıyla asla değişmeyeceği ve güvenlik riskinin ortadan kalkacağını gösterir. Bu nedenle para doğru hesaba eklenir.

String nesnesine işaret eden düzinelerce, yüzlerce veya binlerce thread olabilir ve hepsini bunu okuyabilir. İstedikleri zaman tüm değişkenler o nesneyi kullanıyor olsa da hiçbiri onu değiştiremez. Yani hepsi güvenli bir şekilde kullanabilirler.

Java neden çoklu kalıtımı desteklemez açıklayın? (5 PUAN)

Karmaşıklığı azaltmak ve dili basitleştirmek için, Java'da birden fazla kalıtım desteklenmez.



Şekil 7: C sınıfı A ve B sınıfından kalıtım alır.

A, B ve C'nin 3 adet sınıf olduğunu kabul ettiğimizde C sınıfı A ve B sınıfından kalıtım almaktadır. A ve B sınıfları aynı yönteme sahip ise ve sınıf nesnesinden A yada B'yi çağırırsak belirsizlik oluşur.

```
public class A{

    public static void print(){

        System.out.print(" A sınıfı methodu" );

    }

}
```

```
public class B{

    public static void print(){

        System.out.print(" B sınıfı methodu" );

    }

}
```

Şekil 8: A ve B sınıfına ait methodlar

C sınıfı içerisinde print() yöntemini çağırdığımızda bu yöntemin A'nın mı B'nin mi içerisinden çağırılacağı belli olmaz. Bundan dolayı derleme zamanı hataları çalışma zamanı hatalarından daha iyi olduğu için,2 sınıfı devraldığı için derleme zamanı hatası oluşur. Yani aynı yönteme veya farklı bir yönteme sahip olmak da yine derleme zamanı hatasına neden olacaktır. Bu sebeple Java çoklu kalıtımı desteklemez.

Build Tool nedir? Java ekosistemindeki build toolar neler açıklayın? (5 PUAN)

Java oluşturma araçları, geliştirme için ihtiyaç duyduğunuz temel araçlardır. Bir derleme aracı olmadan kod yazamazsınız, çünkü bir IDE'de kod yazdıktan sonra, kodu derlemek, paketleri oluşturmak ve kaynak koddan yürütülebilir bir Java uygulaması oluşturmak için bir Java derleme aracı gerekir.

Build toollar kullanım amaçları proje sürecini otomatikleştirmektir proje süreci:

- **Derleme** : Kaynak kodunu makine koduna derlemek
- **Bağımlılık yönetimi** : Gerekli üçüncü taraf entegrasyonlarını belirleme ve düzeltme eki uygulama
- **Otomatik testler** : Yürütme testleri ve raporlama hatası
- **Dağıtım için paketleme uygulaması** : Kaynak kodunu sunuculara dağıtım için hazırlar

Java ekosisteminde Maven,Grandle ve Ant olmak üzere 3 adet build tool bulunmaktadır.

Maven

Proje oluşturma sürecini otamatikleştirir.XML tabanlı bir oluşturma aracı ve bağımlılık yöneticisidir. Varsayılan deposu Maven Merkez Deposu'dur ve açık kaynaklı bileşenlerden oluşur. Proje Nesne Modeli (POM) dosyaları tarafından tanımlanır. Bu POM.xml dosyaları, projenin bağımlılıklarını, eklentilerini, özelliklerini ve yapılandırma verilerini içerir.

POM dosyaları → Her projenin bir pom dosyası olmalıdır. POM dosyasında Maven komutları yürütebilmesi için kaynaklar yazılır ve Maven komutu run ettiğimizde Maven komutu POM'da açıklanan kaynaklar üzerinde yürütür.

- Modelversion → POM modelinin sürümünü içerir.
- artifactId→ Oluşturmakta olduğumuz projenin adını içerir.
- versionId→ Sürüm numarasını içerir
- dependencies→ Proje bağımlılıklarını içerir.
- properties →Java derleyici özelliklerini içerir.

Grandle

Gradle, hem Maven hem Ant'ı güzel bir şekilde birleştirir.Proje yapılandırmasında XML kullanan Maven'den farklı olarak Groovy veya Kotlin programlama diline dayalı etki alanına özgü bir dil kullanır. Çok projeli yapılar düşünülerek tasarlanmıştır. Maven'e kıyasla daha hızlıdır.

ANT

Java uygulamaları oluşturmak için oluşturulmuş ilk oluşturma aracı. Java'da uygulanır ve derleme betikleri XML'de yazılır.

Target → oluşturma sürecinin bir bölümünü (veya tamamını) gerçekleştirmek için yürütülecek bir görevler dizisi

- clean → derlenmiş sınıf dosyalarını silmek, oluşturulan kaynak kodunu silmek veya yalnızca tüm yapı çıktı dizinini tamamen silmek.
- Compile →derleme
- Jar→Derlenmişsınıflardan jar dosyası oluşturur.
- Test→Tüm birim testleri çalıştırır.
- Javadoc→JavaDoc yprumları oluşturmak için kullanılır.