Nesne Tabanlı Programlama

GİRİŞ

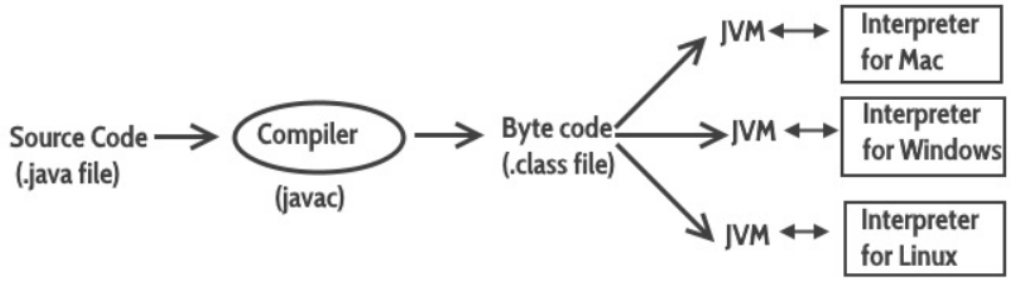
DR. ŞAFAK KAYIKÇI

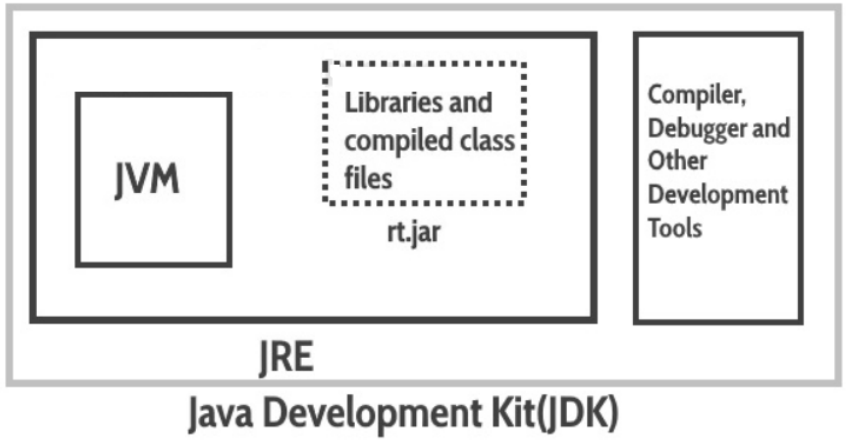
Java

- 1991 yılında Sun Microsystems Inc tarafından geliştirilmiştir ve daha sonra Oracle Corporation tarafından satın alınmıştır.
- James Gosling ve Patrick Naughton tarafından geliştirilmiştir.
- Modüler programlar ve yeniden kullanılabilir kodlar oluşturmaya yardımcı olan, yazması, derlemesi ve hata ayıklaması kolay olan bir programlama dilidir.

Java Terminolojisi

Java Sanal Makinesi (JVM) : JVM'nin birincil işlevi derleyici tarafından üretilen bayt kodunu yürütmektir. Her işletim sistemi farklı JVM'ye sahiptir, ancak bayt kodu çalıştırıldıktan sonra ürettikleri çıktı tüm işletim sistemlerinde aynıdır . Bu nedenle java'yı platformdan bağımsız dil olarak adlandırılır. (WORA : Write Once Run Anywhere)





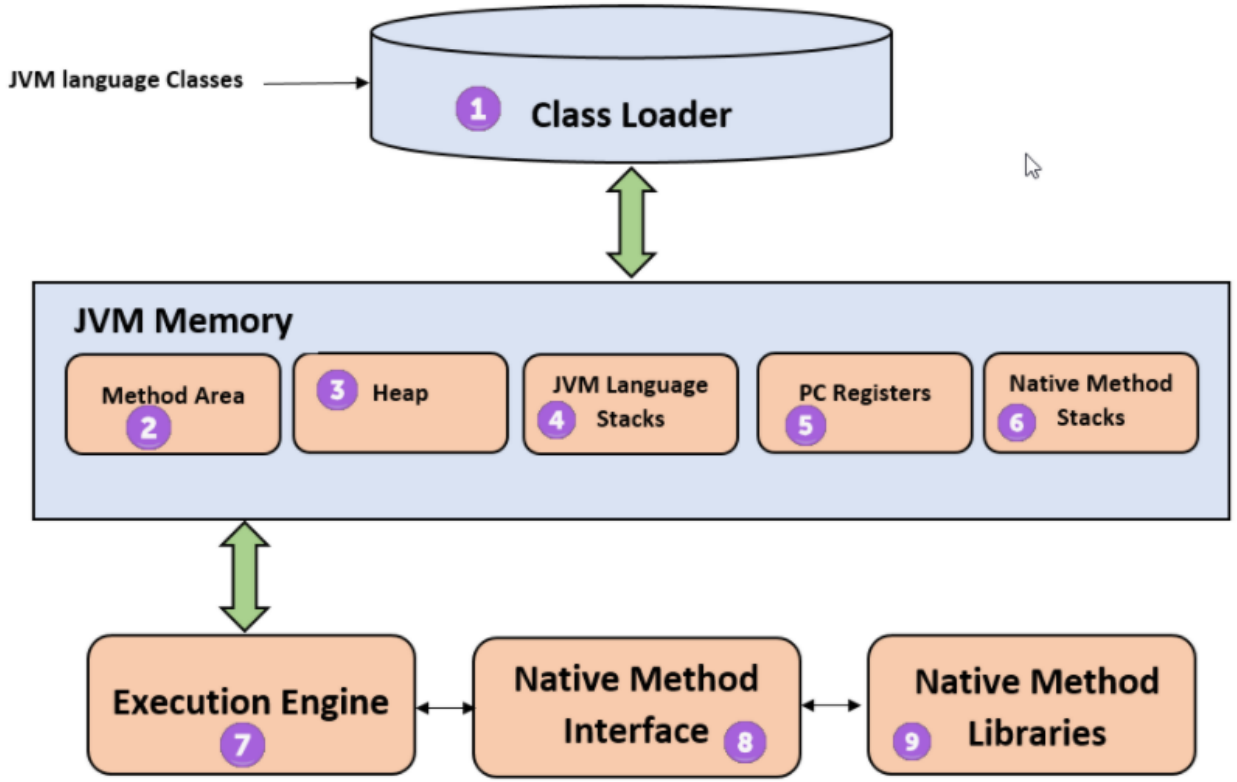
JDK'nın **javac** derleyicisi java kaynak kodunu bayt kodu halinde derler, böylece JVM tarafından yürütülebilir. Bayt kodu, derleyici tarafından bir .class dosyasına kaydedilir.

Java Terminolojisi

Java Geliştirme Kiti (JDK) : JRE (Java Runtime Environment), derleyiciler ve JavaDoc, Java hata ayıklayıcı gibi çeşitli araçları içeren eksiksiz bir java geliştirme kitidir. Java programı oluşturmak, derlemek ve çalıştırmak için bilgisayarınızda JDK'nın kurulu olması gerekir.

Java Runtime Environment (JRE) : JRE, JDK'nın bir parçasıdır ve bu, JDK'nın JRE'yi içerdiği anlamına gelir. Sisteminize JRE yüklediğinizde, bir java programını çalıştırabilirsiniz ancak onu derleyemezsiniz. JRE, JVM, tarayıcı eklentileri ve uygulama desteği içerir. Bilgisayarınızda yalnızca bir java programını çalıştırmanız gerektiğinde, yalnızca JRE'ye ihtiyacınız olacaktır.

JVM Mimarisi



JVM Mimarisi

1) ClassLoader: Sınıf yükleyici, sınıf dosyalarını yüklemek için kullanılan bir alt sistemdir. Üç ana işlevi yerine getirir. Yükleme, Bağlama ve Başlatma.

2) Method Area: JVM Yöntem Alanı, meta veriler, sabit çalışma zamanı havuzu ve yöntemlerin kodu gibi sınıf yapılarını depolar.

3) Heap: Tüm Nesneler, ilgili örnek değişkenleri ve diziler burada saklanır. Bu bellek yaygındır ve birden çok iş parçacığı arasında paylaşılır.

4) JVM Language Stacks: Java dil Yığınları yerel değişkenleri ve kısmi sonuçları depolar. Her iş parçacığı,kendi JVM yığınına sahiptir. Bir yöntem her çağrıldığında yeni bir çerçeve oluşturulur ve yöntem çağırma işlemi tamamlandığında silinir.

5) PC Registers :PC kaydı şu anda yürütülmekte olan Java sanal makine talimatının adresini depolar. Java'da her iş parçacığının ayrı bir bilgisayar kaydı vardır.

6) Native Method Stacks: Yerel yöntem yığınları, yerel kodun talimatını yerel kitaplığa bağlıdır. Java yerine başka bir dilde yazılmıştır.

7) Execution Engine: Donanımı, yazılımı veya tüm sistemleri test etmek için kullanılan bir yazılım türüdür. Test yürütme motoru, test edilen ürünle ilgili hiçbir bilgi taşımaz.

8) Native Method interface : Yerel Yöntem arayüzü bir programlama çerçevesidir. Bir JVM'de çalışan Java kodunun kütüphaneler ve yerel uygulamalar tarafından çağrılmasına izin verir.

9) Native Method Libraries : Yerel Kitaplıklar, Yürütme Motoru tarafından ihtiyaç duyulan Yerel Kitaplıklar (C, C ++) koleksiyonudur.

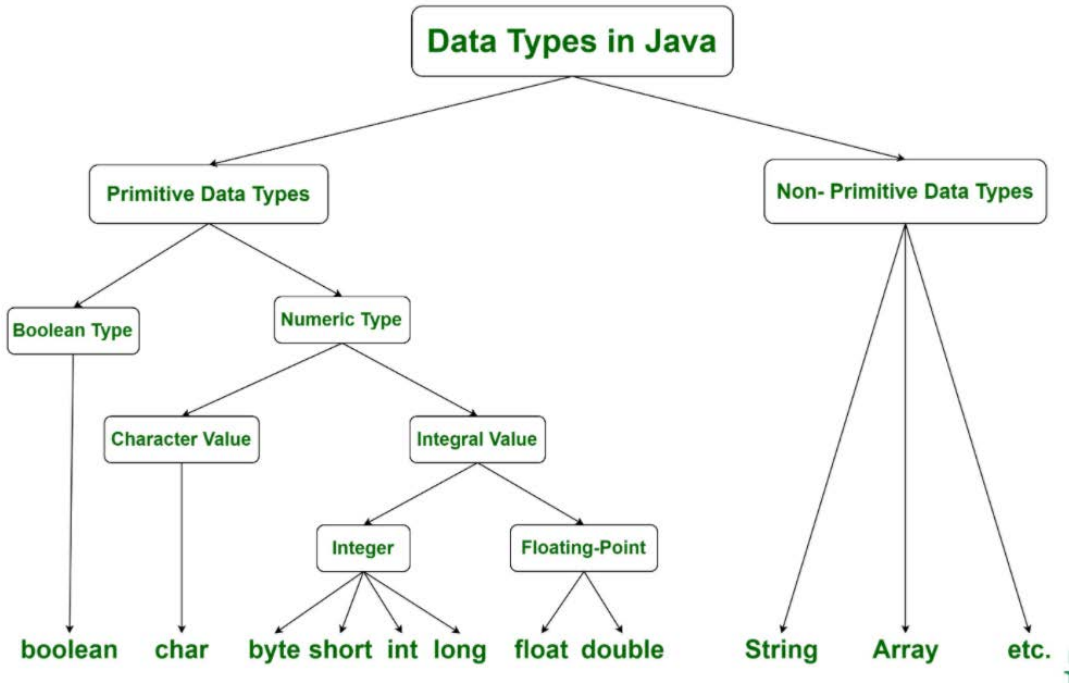
Java Özellikleri

- Platformdan bağımsız
- Nesne Tabanlı

  Soyutlama (Abstraction)
  Kapsülleme (Encapsulation)
  Kalıtım (Inheritance)
  Çok Biçimcilik (Polimorfizm)

- Sağlam (Robust)
- Güvenli (Secure)
- Dağıtık (Distributed) : RMI (Remote Method Invocation), EJB (Enterprise Java Beans) vb.
- Multithread
- Taşınabilir (Portable)

Veri Türleri (data types)



TYPE	DESCRIPTION	DEFAULT	SIZE	EXAMPLE LITERALS	RANGE OF VALUES
boolean	true or false	false	1 bit	true, false	true, false
byte	twos complement integer	0	8 bits	(none)	-128 to 127
char	unicode character	\u0000	16 bits	'a', '\u0041', '\101', '\\', '\'','\n',' β'	character representation of ASCII values 0 to 255
short	twos complement integer	0	16 bits	(none)	-32,768 to 32,767
int	twos complement integer	0	32 bits	-2, -1, 0, 1, 2	-2,147,483,648 to 2,147,483,647
long	twos complement integer	0	64 bits	-2L, -1L, 0L, 1L, 2L	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	IEEE 754 floating point	0.0	32 bits	1.23e100f, -1.23e-100f, .3f, 3.14F	upto 7 decimal digits
double	IEEE 754 floating point	0.0	64 bits	1.23456e300d, -1.23456e-300d, 1e1d	upto 16 decimal digits

Değişken adlandırma kuralları (naming conventions)

1. Değişken isimlendirmesi beyaz boşluklar içeremez, örneğin: int num ber = 100; geçersiz çünkü değişken adında boşluk var.
2. Değişken adı $ ve _ gibi özel karakterlerle başlayabilir.
3. Java kodlama standartlarına göre değişken adı küçük harfle başlamalıdır, örneğin int numarası; Birden fazla kelime içeren uzun değişken isimleri için : int smallNumber; int bigNumber; (ikinci kelimeye büyük harfle başlanır).
4. Değişken adları Java'da büyük / küçük harfe duyarlıdır.

Değişken Türleri (Variable Types)

1. Statik (veya sınıf) değişken : Sınıfla ilişkilendirilir ve sınıfın tüm örnekleri için ortaktır. Statik değişkenlere nesneleri kullanmadan erişebilir.
2. Instance (örnek) değişken: Her sınıf nesnelerinin kendi örnek değişken kopyası vardır.
3. Yerel değişken: Bu değişkenler, sınıfın yöntemi (metot, yordam) içinde bildirilir. Kapsamları yöntemle sınırlıdır. Metot dışında erişelemez.

Operatörler

Operation Types	Operators	Examples
Arithmetic Operations	+, -, *, /, %, ++, --	A+B, A-B
Relational Operations	==, !=, >, <, >=, <=	A==B, AI=B
Logical Operations	&&, || , !	(A==B) && (A<B)
Assignment Operations	=, +=, -=, *=, /=, %=	B=10, A+=20
Ternary Operations	(condition)? value if true : Value if false	X=(A<B)? 10:20
Bitwise Operations	&, |, ~, ^	A&B, A | B

Kontrol İfadeleri

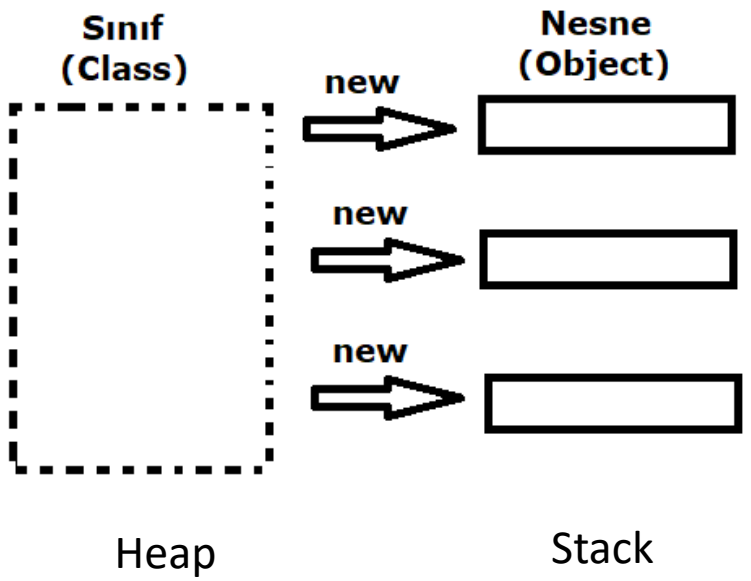
- If-else
- Switch-Case
- For loop
- while loop
- do-while loop
- Continue statement
- break statement

Nesne Tabanlı Programlama

SINIF-NESNE KAVRAMLARI

DR. ŞAFAK KAYIKÇI

Sınıf - Nesne



Nesne : Ev
Durumu (State): Adres, Renk, Alan
Davranışı (Behavior): Açık kapı, kapı kapalı

Nesne : Araba
Durumu (State): Renk, Marka, Ağırlık, Model
Davranışı (Behavior): Kır, Hızlandır, Yavaşla, Vites değiştirme

Method Overloading (Aşırı Yükleme)

Metot aşırı yükleme, argüman sayısını değiştirerek veya argümanların veri türünü değiştirerek yapılabilir .

İki veya daha fazla yöntem aynı ada ve aynı parametre listesine sahipse ancak dönüş türü farklıysa aşırı yüklenemez.

Not: Aşırı yüklenmiş yöntemin farklı erişim değiştiricileri olabilir ve yöntem aşırı yüklemesinde herhangi bir önemi yoktur.

Yöntem aşırı yüklemesinin iki farklı yolu vardır.

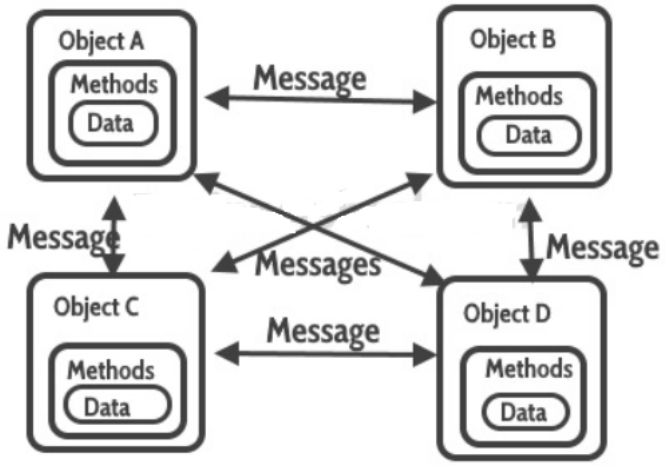
1. Farklı veri türü bağımsız değişkenler
2. Farklı sayıda argüman

Nesnelerin Özellikleri

Abstraction (Soyutlama) : Soyutlama, yalnızca "ilgili" verileri gösterdiğiniz ve bir nesnenin gereksiz ayrıntılarını kullanıcıdan "gizlediğiniz" bir süreçtir.

Encapsulation (Kapsülleme) : Kapsülleme, basitçe nesne durumunu (alanları) ve davranışı (yöntemleri) birbirine bağlamak anlamına gelir.

Message passing (İleti aktarımı): Tek başına tek bir nesne çok yararlı olmayabilir. Bir uygulama birçok nesne içerir. Bir nesne başka bir nesneyle, o nesnede yöntemler çağırarak etkileşime girer. **Method Invocation** (Yöntem Çağrısı) olarak da anılır .



Constructor (Yapılandırıcı)

Yapılandırıcı metodlar, nesneyi oluşturduğumuz anda çalıştıran metodlardır.

Yapılandırıcılar, bellekte yeni bir nesne oluştururlar ve her çağrılışlarında farklı farklı nesneler oluşturacaklardır.

Yapılandırıcıların erişim belirleyicisi public olmalıdır.

Yapılandırıcı ismi, o an çalışma yapılan sınıfın adıyla aynı olmalıdır.

Yapılandırıcı metod çağırılırken new anahtar sözcüğü kullanılmalıdır.

this kelimesi

Java'da **this**, bir sınıfın mevcut nesnesine atıfta bulunmak için kullanılan bir anahtar sözcüktür.

Instance (örnek) ve Local (yerel) değişkenler için aynı değişken adına sahip olunduğunda çıkabiliecek karışıklığı çözmek için kullanılır.

Ayrıca kalıtımda da ata sınıf ile mevcut sınıf değişkenlerini ayırt etmek içinde kullanılır.

Nesne Tabanlı Programlama

KALITIM - POLYMORPHISM

DR. ŞAFAK KAYIKÇI

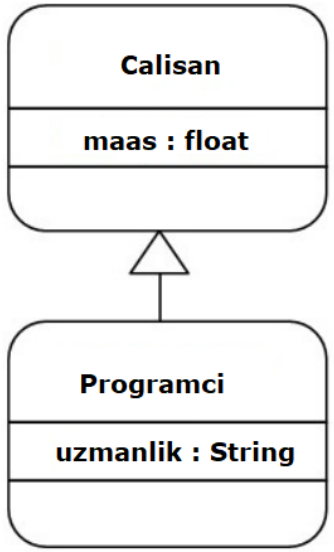
Inheritance (Kalıtım)

Bir nesnenin, bir **ata** nesneden tüm özelliklerini ve davranışlarını elde ettiği bir mekanizmadır.

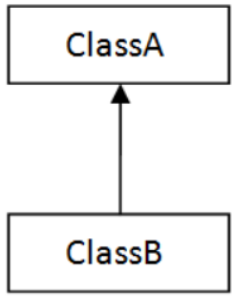
Mevcut bir sınıftan miras aldığınızda, üst sınıfın yöntemlerini ve alanlarını yeniden kullanabilirsiniz. Ayrıca, mevcut sınıfınıza yeni yöntemler ve alanlar da ekleyebilirsiniz.

Kalıtım, *ebeveyn-çocuk* (parent-child) ilişkisi olarak da bilinen **IS-A** ilişkisini temsil eder.

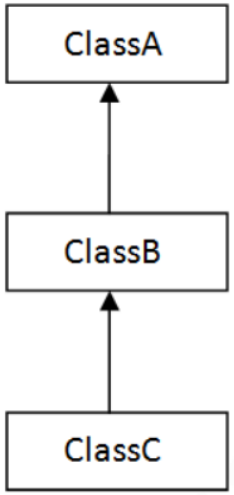
```
class  Subclass  extends Superclass    {
   // yöntemler ve alanlar
}
```



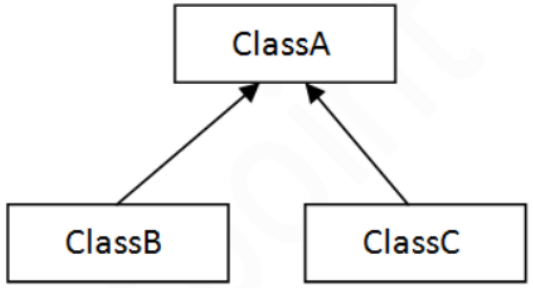
Kalıtım Türleri



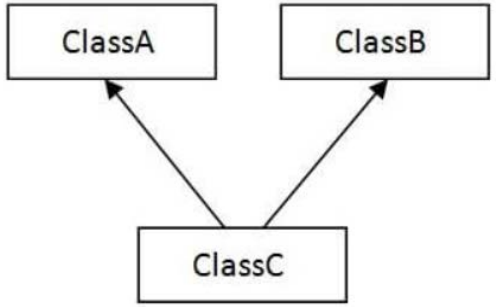
1) Single



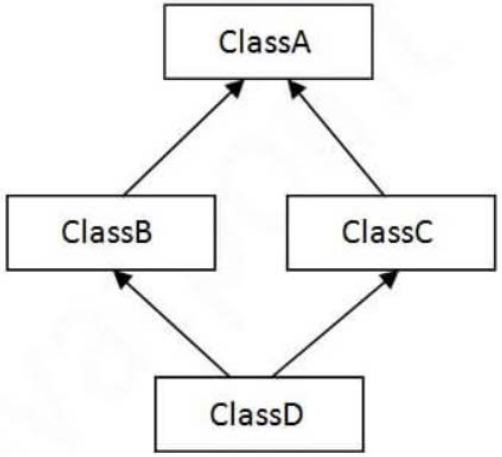
2) Multilevel



3) Hierarchical



4) Multiple



5) Hybrid

!! Java'da multiple inheritance yoktur. !!

Neden Java’da çoklu kalıtım yok ?

Karmaşıklığı azaltmak ve dili basitleştirmek için, java'da çoklu kalıtım desteklenmez.

Compile Time (derleme zamanı) hataları, Run Time (çalışma zamanı) hatalarından daha iyi olduğundan, iki sınıftan türetirseniz Java derleme zamanı hatası oluşturur.

```
Class A {
     void msg () {System.out.println ( "Merhaba" );}
 }

Class B {
     void msg () {System.out.println ( "Hoş Geldiniz" );}
 }

 C extends A, B { // Derleme hatası (kabul olduğunu varsayalım)
    public static void main (String args []) {
      C obj = yeni C ();
      obj.msg (); // Şimdi hangi msg () yöntemi çağrılacak?
    }
  }
```

Overriding (Metot Ezme)

Alt sınıfın (subclass), ata sınıftan (parent class) gelen bir aynı isimli metodu ezerek, kendisinde uyarlamasıdır.

Overloading (aşırı yükleme) ile farkları :

- Overloading aynı sınıf içinde gerçekleştirilir. Overriding, IS-A (kalıtım) ilişkisi olan iki sınıfta gerçekleşir .
- Overloading durumunda parametre *farklı* olmalıdır. Overriding durumunda, parametre *aynı* olmalıdır.
- Overloading derleme zamanı ile ilgilidir. Overriding çalışma zamanı ile ilgilidir.

super keyword

Java'da **super** nahtar kelimesi üst sınıf nesnesi başvurmak için kullanılan bir referans değişkendir.

Her alt sınıf oluşturulduğunda, super referans değişkeni tarafından başvurulan bir üst sınıf örneği implicit (örtülü) olarak oluşturulur.

super : üst sınıf değişkenini belirtmek için, üst sınıf yöntemini çağırmak için kullanılabilir ve üst sınıf yapılandırıcısını çağırmak için kullanılabilir.

```
class Hayvan {
  String renk = "beyaz" ;
  }

class Kopek extends Hayvan {
  String renk = "siyah" ;
  void renkGoster () {
      System.out.println (renk); // Kopek sınıfının rengini yazdırır
      System.out.println ( super. renk); // Hayvan sınıfının rengini
yazdırır
   }
  }
```

```
class Test {
  public static void main (String args [])
{
  Kopek d = yeni Kopek ();
  d. renkGoster ();
 }
}
```

final keyword

Java'daki **final** kelimesi kullanıcıyı kısıtlamak için kullanılır. Final şunlar olabilir: değişken, yöntem, sınıf. Constructor (yapıcılar) final olmaz.

değişken (variable)

```
class Bisiklet{
  final int limit = 90 ; // final variable
  void run () {
    limit = 400 ;
  }
 public static void main (String args []) {
    Bisiklet obj = new Bisiklet ();
     obj.run ();
  }
}
```

yöntem (method)

```
class Bisiklet{
  final void run() {System.out.println("gidiyor");}

 class Bianchi extends Bisiklet {
   void run () {System.out.println ("" );}

    public static void main (String args []) {
      Bianchi b = new Bianchi ();
      b.run ();
    }
  }
```

sınıf (class)

```
final class Bisiklet{}

 class Bianchi extends Bisiklet {
  void run {System.out.println("gidiyor"); }

   public static void main(String args[]){
     Bianchi b = new Bianchi ();
      b.run ();
   }
}
```

Output:Compile Time Error

Aggregation (Münasebet)

Bir sınıfın bir sınıfa referansı varsa, bu aggregation olarak bilinir.

Aggregation, **HAS-A** ilişkisini temsil eder.

Kodun yeniden kullanılabilirliği için kullanılır (Reusability).

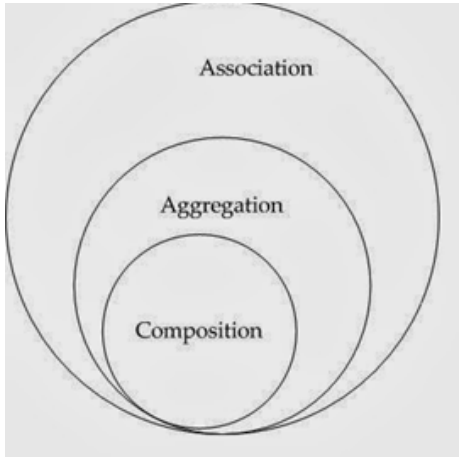
```
class Calisan {
  int id;
  Adres adresi; // Adres bir sınıftır
 }
```

İlişkiler

Inheritance>Composition>Aggregation>Association

- **Association (Birliktelik)**: **uses a**

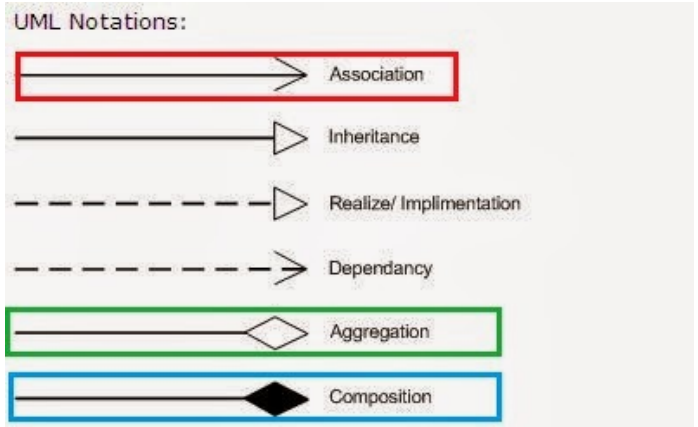
  Ex: a Class Man ***uses a*** Class Pen (Pen is still there when man die)



- **Aggregation (Münesabet)**: **has a**

  Ex: a Class Man ***has a*** Class Car (Car is still there when Man die)

- **Composition (Oluşum): owns a**
  Ex: a Class Man ***owns a*** Class Heart (When Man die, Heart die)



- **Inheritance (Kalıtım): is a**
  Ex: a Class Man ***is a*** Class Human (Man is a Human)

Polymorphism (Çok Biçimcilik)

"Poli" kelimesi birçok anlamına gelir ve "morf" formlar anlamına gelir. Yani polimorfizm, birçok form anlamına gelir.

Polymorphism bir nesnenin farklı bir nesne gibi davranmasıdır.

Static (Early) -Dynamic (Late) Binding

Erken Bağlama — Early Binding – Static Binding

Compile time (derleme zamanı) aşamasında oluşur. static, private ve final metotlar bu şekilde bağlanır. Çünkü bu metotlar ezilemezler ve sınıfın cinsi derleme zamanında belirlenir. Metot aşırı yükleme de (overloading) de bu duruma örnektir.

Geç Bağlama — Late Binding – Dynamic Binding

Polymorphism oluştuğu anda, yani run time (çalışma zamanı) sırasında nesne örneğinin bağlanacağı nesne türünün belirlenmesi ve buna uygun işlemlerin tamamlası sürecinde yapılan işlemlerdir. Çok biçimlilik olmadan geç bağlamadan bahsedilemez. Metot ezme (overriding) de bu duruma örnektir.

instanceof

Java'da **instanceof** operatörü, nesnenin belirtilen bir türün (sınıf veya alt sınıf veya arabirim) bir örneği olup olmadığını test etmek için kullanılır.

true yada **false** döndürür. null değeri olan herhangi bir değişkenle uygulanırsa false döner.

```
class  Hayvan { }
class  Kopek extends Hayvan {
      public static void  main (String args []) {
            Kopek k = new   Kopek ();
            System.out.println (k instanceof  Hayvan);
      }
}
```

downcasting

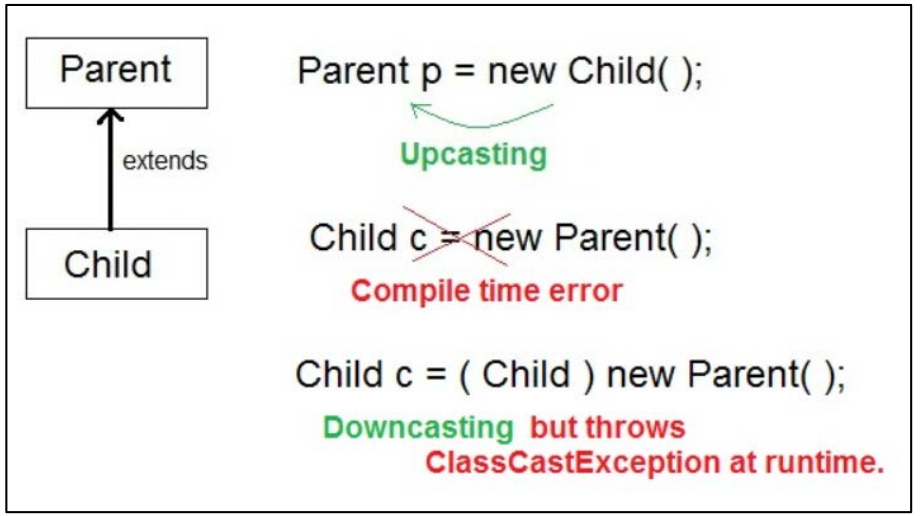
Alt sınıf türü, üst sınıfının nesnesine atıfta bulunduğunda, **downcasting** olarak bilinir.

- Direkt olarak yapılırsa compiler Compilation hatası alınır.

```
Kopek k = new  Hayvan (); //Derleme Hatası
```

- Typecasting ile yapılırsa, çalışma zamanında ClassCastException alınır.

```
Kopek k = (Köpek)  new   Hayvan ();
// Başarıyla derlenir ancak ClassCastException çalışma zamanında alınır
```



çöp toplayıcı (garbage collector)

Java çöp toplama, kullanılmayan nesneler tarafından işgal edilen kullanılmayan **belleği serbest bırakma işlemidir** . Bu işlem, bellek yönetimi için gerekli olduğu için JVM tarafından otomatik olarak yapılır.

JVM'de programlar **heap** üzerinde nesneler oluşturulur. Sonunda bazı nesnelere artık ihtiyaç duyulmayacaktır. Bir nesneye herhangi bir referans olmadığında, o nesneye artık ihtiyaç duyulmadığı varsayılır ve nesnenin kapladığı bellek serbest bırakılır. Bu işleme **garbage collection** (çöp toplama) denir .

System.gc () yöntemini çağırarak JVM'den çöp toplama talebinde bulunabilir. Çöp toplama işleminden önce, garbage collector tarafından **finalize()** methodu çağrılır.

1

```
Kopek kopek = new Kopek();
kopek = null; // çöp
toplama için hazır
```

2

```
Kopek kopek = new Kopek();
Kopek kopek2 = new Kopek();
kopek2 = kopek; // referans değişimi
```

Nesne Tabanlı Programlama

ERİŞİM SEVİYLERİ

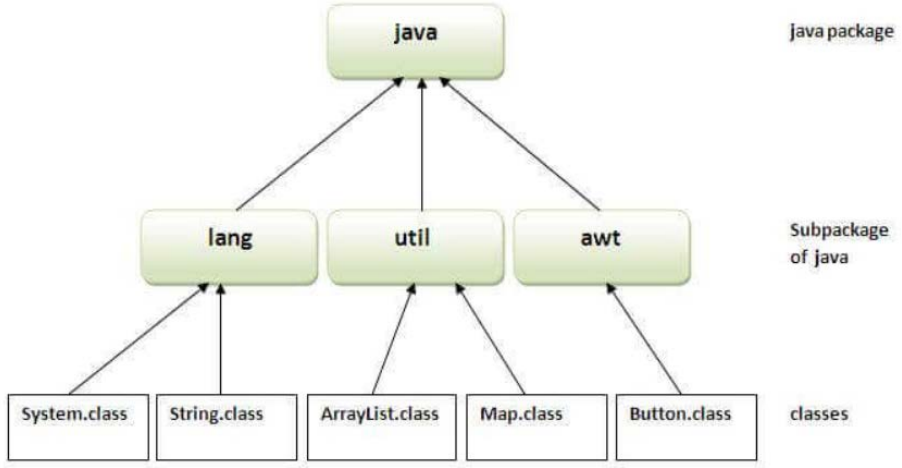
DR. ŞAFAK KAYIKÇI

java package

package (paket) ilgili sınıfların bir koleksiyonudur. Java, ilgili sınıfları, arayüzleri ve alt paketleri gruplamak, çakışmalarını önlemek ve sınıf, arayüz ve enumariton vb. erişimlerini kontrol etmek için kullanılır. **package ifadesi, import ifadesinden bile önce programdaki ilk ifade olmalıdır.**

1. Built-in Package: math, util, lang, i/o vb.

2. User-defined-package: Kullanıcı tarafından projelerinin sınıflarını ve arayüzlerini kategorize etmek için oluşturulan kullanıcı tanımlı paketler olarak bilinir.



1. import package.*;
2. import package.classname;
3. fully qualified name.

static import

Bir sınıfın statik üyesini içe aktarmak için kullanılır.

```
import static java.lang.Math.sqrt;    //importing static method sqrt of Math class
```

```
import static java.lang.Math.*;    //importing all static member of Math class
```

```
import static java.lang.Math.*;
public class Test
{
    public static void main(String[] args)
    {
        System.out.println(sqrt(144));
    }
}
```

Erişim belirleyiciler (Access Modifiers)

Java'daki erişim beliryeciler, bir özelliğin, yöntemin, yapıcının veya sınıfın erişilebilirliğini veya kapsamını belirtir.

Private : Private değiştiricinin erişim seviyesi yalnızca sınıfın içindedir. Sınıf dışından erişilemez.

Default (**friendly):** Varsayılan bir değiştiricinin erişim düzeyi yalnızca paketin içindedir. Paket dışından erişilemez. Herhangi bir erişim seviyesi belirtmezseniz, bu varsayılan olacaktır.

Protected : Korumalı bir değiştiricinin erişim düzeyi paketin içinde ve alt sınıf aracılığıyla paketin dışındadır. Alt sınıf yapmazsanız paket dışından erişilemez.

Public : Genel değiştiricinin erişim düzeyi her yerdedir. Sınıf içinden, sınıf dışından, paket içinden ve paket dışından erişilebilir.

Bunların dışında static, abstract, synchronized, native, volatile, transient vb. gibi birçok non-access modifiers vardır.

Kapsamları

	Class	Package	Subclass (same package)	Subclass (diff package)	Outside package
public	Yes	Yes	Yes	Yes	Yes
protected	Yes	Yes	Yes	Yes	No
default	Yes	Yes	Yes	No	No
private	Yes	No	No	No	No

Encapsulation (Kapsülleme)

Kapsülleme, kodu ve verileri sarmalayarak *tek bir birim* halinde getirmektir. Örn: çeşitli ilaçların karıştırıldığı bir kapsül.

Sınıfın tüm veri üyelerini **private** yaparak Java'da tamamen kapsüllenmiş bir sınıf oluşturabilir. Daha sonra, içindeki verileri ayarlamak ve almak için **getter** ve **setter** yöntemleri kullanabilir. Örn : **Java Bean**

Encapsulation Avantajları

- **Veriler üzerinde kontrol** sağlar . Yalnızca 100'den büyük olması gereken id değerini ayarlamak için yada negatif girişi engelleme kontrolünü setter yönteminin içine yazabilirsiniz.
- **veri gizlemeyi (data hiding)** sağlamanın bir yoludur çünkü diğer sınıflar private verilere erişemeyecektir.
- **test edilmesi kolaydır** . Bu nedenle, unit test için daha iyidir.

Nesne Tabanlı Programlama

SOYUT SINIF - ARAYÜZ

DR. ŞAFAK KAYIKÇI

abstract class (soyut sınıf)

Soyutlama (abstraction) , uygulama ayrıntılarını gizleme ve kullanıcıya yalnızca işlevselliği gösterme işlemidir.

Soyut sınıflar genellikle iki veya daha fazla alt sınıfın farklı uygulamalar yoluyla farklı şekillerde benzer bir şey yapmasının beklendiği durumlarda açıklanır. Bu alt sınıflar, aynı soyut sınıfını genişletir ve soyut yöntemler için farklı uygulamalar sağlar.

Soyut sınıflar, nesneye yönelik programlama sınıfı hiyerarşisinin en üstündeki genel davranış türlerini tanımlamak için kullanılır ve soyut sınıfın uygulama ayrıntılarını sağlamak için alt sınıflarını kullanır.

abstract class (soyut sınıf)

- **abstract** anahtar sözcüğü ile kullanılır.
- Soyut sınıflar içerilerinde soyut olmayan metotlar da barındırabilir ancak soyut metotlar sadece soyut sınıflar içerisinde bildirilebilir.
- new ile nesne olarak yaratılamaz (örneklenemez).
- yapıcılara (constructor) ,statik ve final yöntemlere sahip olabilir. Ancak statik ve final yöntemler abstract olamaz.

interface (arayüz)

Interface, Java’da %100 soyutlamayı sağlamak için kullanılan bir kavramdır.

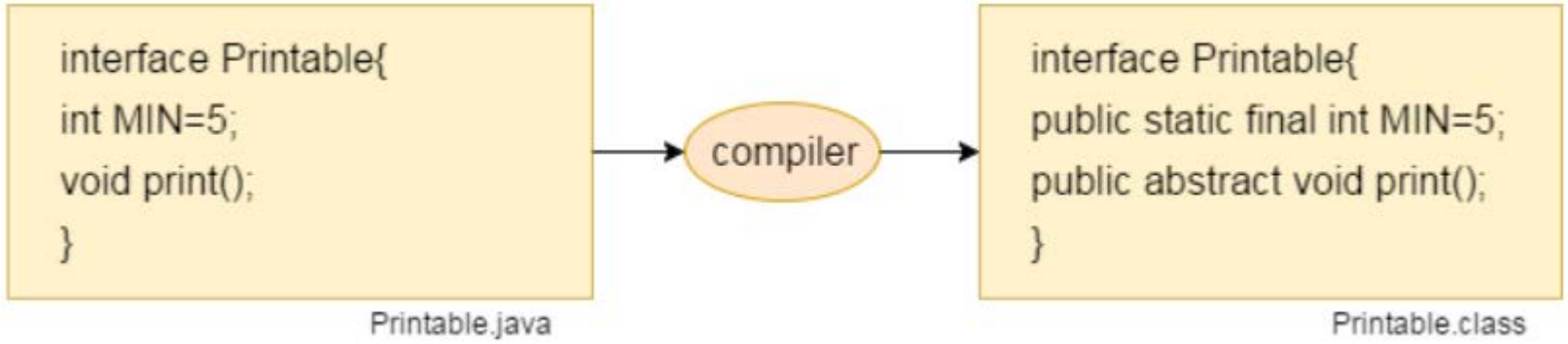
Bir arayüzü uygulamak için **implements** anahtar sözcüğü kullanılır .

- Yalnızca **soyut yöntemlere ve statik alanlara** sahip olabilir. Ancak **Java 8'den** itibaren **default** ve **statik yöntemlere** sahip olabilir ve **Java 9'dan** da **private yöntemlere** sahip olabilir.
- Arayüzler de **IS-A** ilişkisini temsil eder .
- new ile nesne olarak yaratılamaz (örneklenemez).
- Çoklu kalıtımı (multiple inheritance) sağlar.

compiler etkisi

Java derleyicisi, interface methodundan önce public ve abstract anahtar sözcüklerini ekler. Ayrıca, özelliklerden (field) önce public, static ve final sözcüklerini ekler.

Diğer bir deyişle, interface alanları varsayılan olarak public, static ve final olup, metotlar public ve abstract tır.



Çoklu Kalıtım

Multiple Inheritance in Java

Java 8

1- Default Method

```
interface CizimYap{
   void ciz();
   default void msg(){System.out.println("default method");}
   }

class Dikdortgen implements CizimYap{
  public void ciz(){System.out.println("Dikdortgen ciziliyor");}
 }

class TestInterfaceDefault{
  public static void main(String args[]){
        CizimYap  d=new  Dikdortgen();
        d.ciz();
        d.msg();
  }
}
```

2- Static Method

```
interface CizimYap{
   void ciz();
   static int cube(int x){return x*x*x;}
   }

class Dikdortgen implements CizimYap{
  public void ciz(){System.out.println("Dikdortgen ciziliyor");}
 }

class TestInterfaceDefault{
  public static void main(String args[]){
        CizimYap  d=new  Dikdortgen();
        d.ciz();
        System.out.println(CizimYap.cube(3));
  }
}
```

Java 9

```
interface Selamlasma{
   // Default method
   default void msg(){ selamVer(); }

   // Private method
   private void selamVer() {
         System.out.println("Selam.. (private)");
   }

}
public class Demo implements Selamlasma{
   public static void main(String[] args) {
         Demo d = new Demo();
         d.msg();   // default method  cagiriliyor

   }
}
```

Java 9 sürümünde, arayüz içinde private yöntemler eklenebilir. Private yöntemin amacı, arayüzün soyut olmayan yöntemleri arasında bazı görevleri paylaşmaktır.

Abstract Class	Interface
1) Soyut sınıfların **abstract ve non-abstract metotları** olabilir.	Arayüzler sadece **abstract metotları** olur. Java 8 den itibaren, **default ve static metotları** da olabilir.
2) Soyut sınıflar çoklu kalıtım (**multiple inheritance**) desteklemez.	Arayüzler çoklu kalıtım (**multiple inheritance**) destekler.
3) Soyut sınıfların **final, non-final, static ve non-static değişkenleri** olabilir.	Arayüzlerin sadece **static ve final değişkenleri** olabilir.
4) Soyut bir sınıf, bir arayüzü implement edebilir.	Arayüz bir soyut sınıfı implement edemez.
5) Soyut sınıfları belirtmek için **abstract** kelimesi kullanılır.	Arayüzleri belirtmek için **interface** kelimesi kullanılır.
6) Soyur bir sınıf, başka bir sınıftan türetilebilir (extend) ve birden çok arayüzü implement edebilir.	Arayüz sadece başka bir arayüzü extend edebilir.
7) Soyut sınıf **extend** kelimesi ile genişletilebilir.	Arayüz **implements** kelimesi ile uygulanır.
8) Soyut sınıfın private, protected, vb. gibi üyeleri olabilir.	Arayüzün üyeleri varsayılan olarak public'tir.

9)**Example:**
public abstract class Shape{
public abstract void draw();
}
	Example:
	public interface Drawable{
	void draw();
	}

Nested (iç içe) Sınıflar

Java'da, başka bir sınıf içinde bir sınıf tanımlanabilir. Bu sınıflara içiçe (Nested) sınıflar denir.

İki tür içiçe sınıf vardır:

Statik olmayan iç içe sınıf (Inner (Dahili) Class) : Statik olmayan iç içe geçmiş bir sınıf, başka bir sınıf içindeki bir sınıftır. Çevreleyen sınıfın (dış sınıf) üyelerine **erişebilir**. Genellikle **Inner Class** olarak bilinir. İç sınıf dış sınıfın içinde yer aldığından, iç sınıftan nesne oluşturabilmek için ilk önce dış sınıfı oluşturulması gerekir.

Statik iç içe sınıf (Static Nested Class) : Statik bir sınıfın bir başka bir sınıf içerisinde tanımlanmasıdır. Dahili sınıfın aksine, iç içe yerleştirilmiş statik bir sınıf, dış sınıfın üye değişkenlerine **erişemez**. Bunun nedeni, statik iç içe geçmiş sınıfın , dış sınıfın bir örneğini oluşturmasını gerektirmemesidir.

Örnek

```
public class OuterClass {
  int adet;

   public class Inner1 {
      double getAdet(){
           return adet;
        }
  }

   public static class Inner2 {
     double getAdet(){
           return adet;  //erişemez
        }
     }
  }
```

Main Method

1- non-static (Inner)

```
OuterClass out = new OuterClass();
OuterClass.Inner1 ornek1= out.new Inner1();

//yukardakinin aynisi
OuterClass.Inner1 ornek1 = new OuterClass().new Inner1();
```

2- static

```
OuterClass.Inner2 ornek2 = new OuterClass.Inner2();
```

Nesne Tabanlı Programlama

İSTİSNA YÖNETİMİ

DR. ŞAFAK KAYIKÇI

Exception Handling (İstisna Yönetimi)

İstisna, programın normal akışını kesintiye uğratan istenmeyen bir olaydır. Bir istisna oluştuğunda program yürütmesi sona erer. Bu gibi durumlarda, sistem tarafından oluşturulan bir hata mesajı alınır. Java'da istisnalar ele alınarak, sistem tarafından oluşturulmuş bir mesaj yerine kullanıcıya sorun hakkında anlamlı bir mesaj gönderilebilir ve programın davranışı değiştirilerek akışın kesilmemesi sağlanabilir.

Örnekler :

```
int a=50/0;//ArithmeticException
```

```
String s=null;
System.out.println(s.length());//NullPointerException
```

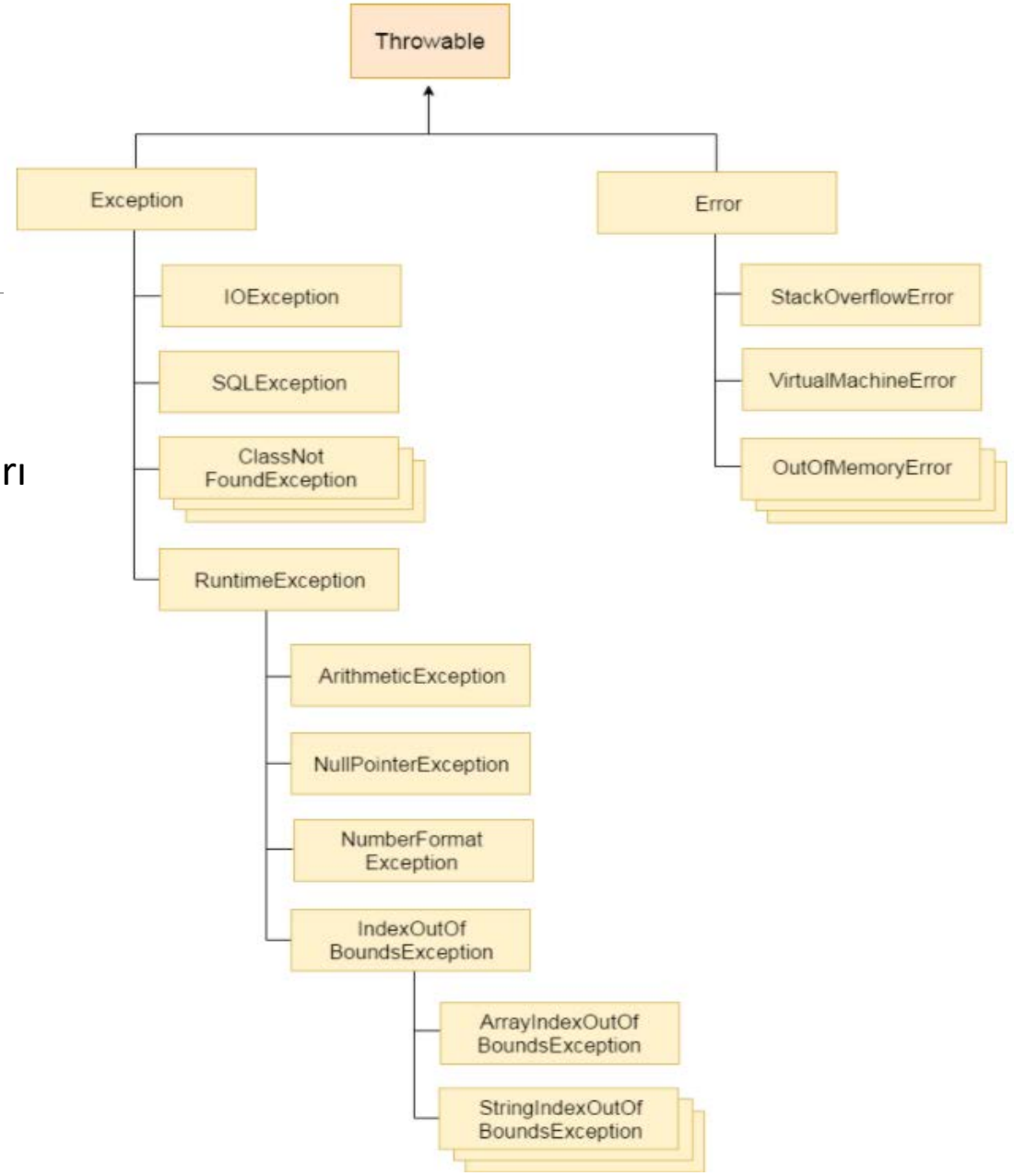
```
String s="abc";
int i=Integer.parseInt(s);//NumberFormatException
```

```
int a[]=new int[5];
a[10]=50; //ArrayIndexOutOfBoundsException
```

```
File f = new File("ornek.txt");
//FileNotFoundException
BufferedReader bf = new BufferedReader( new FileReader( f ) );

//IOException
System.out.println(bf.readLine());
```

Exception Hiyerarşisi

java.lang.Throwable sınıfı ata sınıftır. Exception ve Error sınıfları bu sınıftan türemiştir.



İstisna Türleri

1) Checked Exceptions (Kontrol edilen İstisnalar) : Derleme zamanında (compile time) kontrol edilirler. Örn : IOException, SQLException, ClassNotFoundException, FileNotFoundException ...

2) Unchecked Exceptions (Kontrol edilmemiş İstisnalar) : Çalışma zamanında (run time) ortaya çıkarlar, derleme zamanında kontrol edilmemişlerdir. Bu istisnaları ele almak ve güvenli bir çıkış sağlamak programcının sorumluluğundadır. Örn: ArithmeticException, NullPointerException, ArrayIndexOutOfBoundsException vb.

try-catch-finally

```
try
{
    istisna yaratabilecek program kodu

}
catch ( exception-1 ex-1 )
{
    istisna-1 durumunu ele alacak kod
}
catch ( exception-2 ex-2 )
{
    istisna-2 durumunu ele alacak kod


}
finally
{
    her durumda çalışacak kod

}
```

throw – (istisna fırlat)

throw anahtar sözcüğü kullanılarak kontrollü yada kontrolsüz (checked, unchecked) istisnalar fırlatılabilir. Çoğunlukla programcıların kendilerinin yaratmış olduğu özel (custom) istisnalar için kullanılır.

```
public class TestThrow{
  static void kontrolEt(int yas){
    if(yas<18)
      throw new ArithmeticException("Resit degil");
    else
      System.out.println("Oy kullanabilir");
   }
   public static void main(String args[]){
      kontrolEt(13);
   }
}
```

throws – (istisna fırlatır)

throws anahtar sözcüğü programcıya, bir istisnanın meydana gelebileceği bilgisini verir, böylece programcının istisna işleme kodunu sağlaması ve böylece normal akışın sürdürülebilmesi sağlanır. Kontrol edilen istisnaları işlemek (run time) için kullanılır. **throws** kullanarak tek seferde birden fazla istisna bildirimi yapılabilir.

```
public void myMethod() throws ArithmeticException, NullPointerException
{
  // Statements that might throw an exception
}

public static void main(String args[]) {
  try {
    myMethod();
  }
  catch (ArithmeticException e) {
    // Exception handling statements
  }
  catch (NullPointerException e) {
    // Exception handling statements
  }
}
```

throw - throws

throw	**throws**
throw bir istisnayı açıkça atmak için kullanılır.	throws bir istisnayı bildirmek için kullanılır.
throw dan sonra instance (örnek) gelir.	throws dan sonra sınıf gleir.
throw metodun içinde kullanılır.	throws metodun tanımında (başında) kullanılır.
throw ile birden fazla istisna fırlatılamaz.	throws ile birden fazla istisna bildirilebilir.

Kullanıcı tanımlı istisnalar (custom)

Özel istisna yardımıyla, kendi istisnanıza ve mesajınıza sahip olabilirsiniz. İstisnayı kullanıcı ihtiyacına göre özelleştirmek için kullanılır. Kullanıcı tanımlı istisnalar, Exception sınıfından türetilir.

```
public class TestThrow{
 static void kontrolEt(int yas) throws InvalidAgeException{
   if(yas<18)
     throw new  InvalidAgeException("Resit degil");
   else
      System.out.println("Oy kullanabilir");
  }
  public static void main(String args[]){
     try{
      kontrolEt(13);
     }
     catch(Exception e){
           System.out.println("istisna olustu: "+e);}
  }
}
```

```
class InvalidAgeException extends Exception{
  InvalidAgeException(String s){
  super(s);
   }
}
```

Nesne Tabanlı Programlama

SARMALAMA

DR. ŞAFAK KAYIKÇI

Autoboxing ve Unboxing

Java 1.5 ile birlikte **ilkel türlerin (primitive types)** karşılık gelen **sarmalayıcı (wrapper)** sınıfına otomatik olarak dönüştürülmesine yönelik özel bir özellik getirmiştir.

Autoboxing: İlkel türlerin, karşılık gelen sarmalayıcı sınıflarının nesnesine otomatik olarak dönüştürülmesidir. Örn : int → Integer, long →Long ...

Unboxing: Autoboxing tersidir. Bir sarmalayıcı sınıfındaki bir nesneyi otomatik olarak karşılık gelen ilkel türe dönüştürülmesidir. Integer → int, Double →double

Primitive Data Type	Wrapper Class
byte	Byte
short	Short
int	Integer
long	Long
float	Float
double	Double
boolean	Boolean
char	Character

Autoboxing Örnekler

Case 1: Metot çağrımı

```
class AutoboxingExample1
{
   public static void myMethod(Integer num){
        System.out.println(num);
   }
   public static void main(String[] args) {
       /* passed int (primitive type), it would be
        * converted to Integer object at Runtime
        */
        myMethod(2);
   }
}
```

Case 2: Koleksiyonlarda

```
ArrayList<Integer> arrayList = new ArrayList<Integer>();
arrayList.add(11); //Autoboxing - int primitive to Integer
arrayList.add(22); //Autoboxing
```

Case 3: Atamalarda

```
Integer inum = 3; //Assigning int to Integer: Autoboxing
Long lnum = 32L; //Assigning long to Long: Autoboxing
```

Unboxing Örnekler

Case 1: Metot çağrımı

```
class UnboxingExample1
{
   public static void myMethod(int num){
        System.out.println(num);
   }
   public static void main(String[] args) {

        Integer inum = new Integer(100);

        /* passed Integer wrapper class object, it
         * would be converted to int primitive type
         * at Runtime
         */
        myMethod(inum);
    }
}
```

Case 2: Koleksiyonlarda

```
ArrayList arrayList = new ArrayList()
int num = arrayList.get(0); // unboxing because get method returns an Integer object
```

Case 3: Atamalarda

```
Integer inum = new Integer(5);
int num = inum; //unboxing object to primitive conversion
```

Arkada ne oluyor?

Autoboxing:

Yazdığımız kod:

```
Integer number = 100;
```

yada java 1.5'ten önce bizim yaptığımız

Derleyicinin yaptığı:

```
Integer number = Integer.valueOf(100);
```

Unboxing:

Yazdığımız kod:

```
Integer num2 = new Integer(50);
int inum = num2;
```

Derleyicinin yaptığı:

```
Integer num2 = new Integer(50);
int inum = num2.intValue();
```

Dikkat

Karşılaştırmalar yaparken ilkel tipleri ve nesneleri karıştırmayın. Bu tür karşılaştırmalar için öngörülemeyen sonuçlar alabilirsiniz.

Yapılması daha iyi olan şey: nesneyi nesnelerle (equals () yöntemini kullanarak) ve ilkel tipleri ilkel tiplerle (“==”, “<” vb. Mantıksal operatörler kullanarak) karşılaştırmaktır.

Hatalar

```
Integer a = 1000;
Integer b = 1000;

System.out.println("Esit mi : " + (a==b));
```

```
Boolean b1 = new Boolean("true");
Boolean b2 = new Boolean("true");

System.out.println("Esit mi: "+ (b1==b2));
```

Nesne Tabanlı Programlama

COLLECTIONS (TORBALAR)

DR. ŞAFAK KAYIKÇI

Collections (Torbalar)

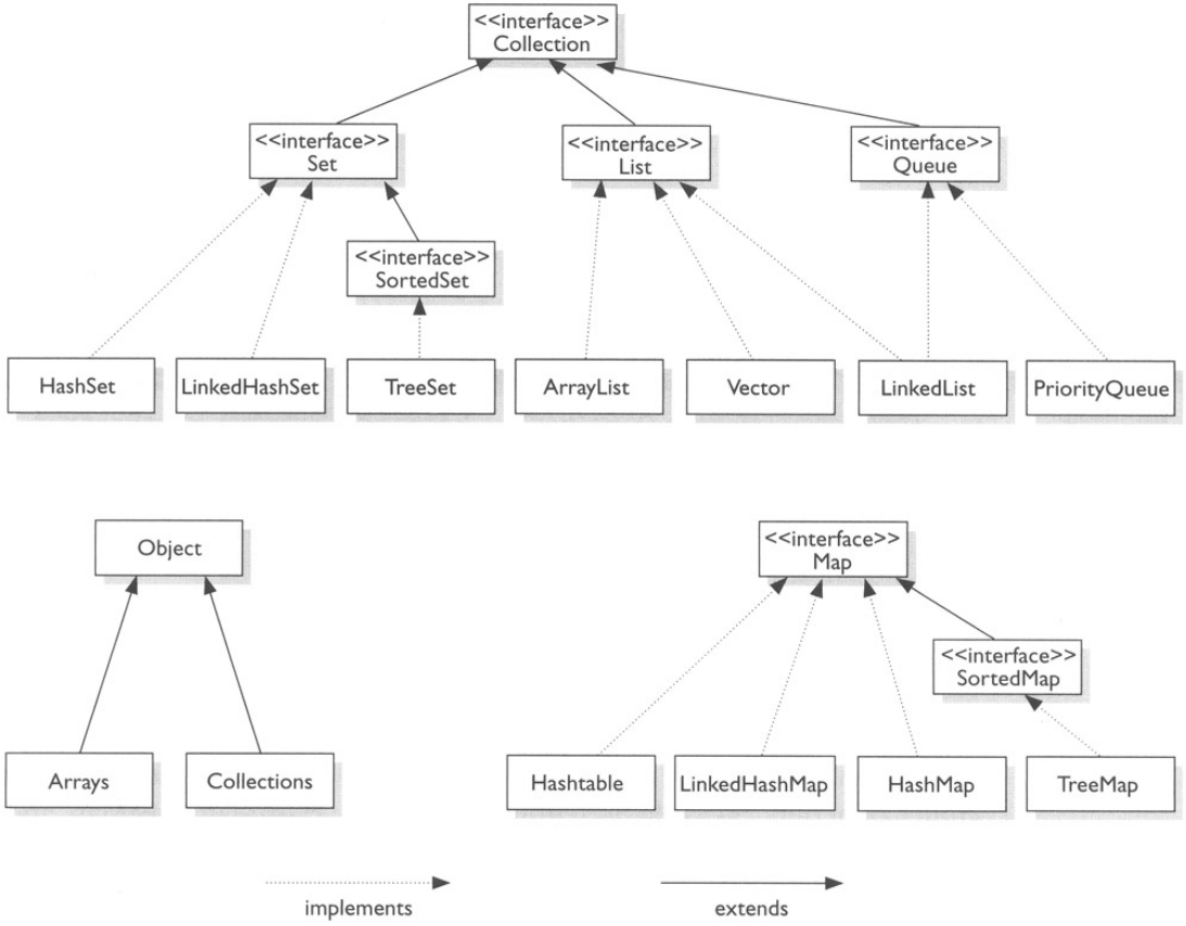
Java'da koleksiyon, birden çok öğeyi tek bir birimde toplayan yapılardır. Verileri depolamak, almak ve işlemek için kullanılır.

Java koleksiyonlar, **java.util** paketinde bulunur. Nesne gruplarını toplamak ve düzenlemek için birçok önemli sınıf ve arayüz sağlar.

Avantajları :

- İçinde hazır bulundurduğu veri yapıları ve algoritmalar ile programlamayı kolaylaştır.
- Veri yapılarının ve algoritmaların uygulamalarını sağlayarak performansı ve hızı artırır.
- Birbirleriyle ilgisiz API'ler aralarında veri geçişini ortak bir dil oluşturarak koleksiyonlarile yapabilirler.
- Üretkenliği artırır

Java Collection Hierarchy



Yaygın Olarak Kullanılan Yöntemleri

Method	Tanım
public boolean add(E e)	Koleksiyona bir öğe ekler
public boolean addAll(Collection<? extends E> c)	Belirtilen koleksiyon öğelerinin tamamını çağıran koleksiyona ekler.
public boolean remove(Object element)	Koleksiyondan bir öğeyi siler.
public boolean removeAll(Collection<?> c)	Belirtilen koleksiyonun tüm öğelerini çağıran koleksiyondan siler.
default boolean removeIf(Predicate<? super E> filter)	Koleksiyonun belirtilen koşulunu karşılayan tüm öğelerini siler.
public boolean retainAll(Collection<?> c)	Belirtilen koleksiyon haricindeki çağıran koleksiyonun tüm öğelerini siler.
public int size()	Koleksiyondaki toplam öğe sayısını döndürür.
public void clear()	Koleksiyonu temizler. (toplam öğe sayısını kaldırır).
public boolean contains(Object element)	Bir elemanı varmı/yokmu arar.
public boolean containsAll(Collection<?> c)	Koleksiyonda belirtilen koleksiyonu aramak için kullanılır.

Collections <Generic> ve Autoboxing

Jenerikler (Generics) , JDK 5.0 ile eklenmiş olup koleksiyonlara **tür güvenliği (type safety)** getirmiş oldu. Daha önceki koleksiyonlar , her türden nesnenin depolayabileceği anlamına gelen **Object sınıfı** referanslarını saklıyordu. Bu durumlarda, bir koleksiyonda uyumsuz türleri depolama ihtimali vardı ve bu da çalışma zamanı uyumsuzluğuna neden olabiliyordu. Bu nedenle, depolanan nesnenin türünü açıkça belirtebileceğiniz Jenerikler tanıtıldı.

Autoboxing, ilkel veri türlerini **sarmalayıcı (Wrapper)** sınıf objelerine dönüştürür. Koleksiyonlar ilkel veri türlerini depolamadığı için (yalnızca referansları depolar), bu nedenle autoboxing ilkel veri türlerinin koleksiyonda saklanmasını sarmalayarak mümkün kılar.

List

List Interface, kullanıcılara sıralanmış bir dizi hazırlama olanağı sunar. Oluşturulan diziler istenilen sınıfta obje saklayabildiği gibi kapasiteleri de önceden belirlenmek zorunda değildir. Bununla beraber bu Interface'i kullanan sınıflar aşağıdaki metodlara sahiptir;

add (E e): Bir objeyi listeye ekler.

add (int index, E e): Bir objeyi listede istenen dizine (index) ekler.

clear: Bütün elemanları siler.

contains: Bir objenin dizi içerisinde olup olmadığını kontrol eder.

get (int index): Dizi içerisinde belirli bir dizindeki objeyi verir.

remove (E e): Belli bir objeyi siler.

remove (int i): Belli bir dizindeki objeyi siler ve dizini günceller.

size: O anda dizide kaç adet eleman olduğunu söyler (add metodu ile eklenmiş).

subList (int from, int to): İki dizin arasındaki elemanlardan yeni bir dizi oluşturur.

set (int index, E element): Belli bir dizindeki objeyi yenisiyle değiştirir.

```
List<String> arrList = new ArrayList<String>();
arrList.add("Osman");
arrList.add("Ayse");
arrList.add(0, "Ozan");
```

```
for(String str : arrList) {
    System.out.println(str);
}
```

Map

Verileri anahtar - değer mantığına göre saklamaktadır. Bir obje Map içerisine eklenirken bu objeyi işaret eden bir anahtar kullanmaktadır. Obje çekilmek istediğinde bu anahtar değeri sorgulanır ve obje hızlı bir şekilde diğer objeler arasından getirilir.

clear: Map içinde bulunan bütün değerleri siler.

containsKey (Object key): Belli bir anahtar daha önceden girilmiş mi sorgular.

containsValue (Object value): Belli bir obje daha önceden girilmiş mi sorgular.

get (Object key): Anahtara karşılık gelen objeyi döndürür.

put (Object key, Object value): Anahtar - değer ikilisini kayıt eder.

remove (Object key): Belli bir anahtara karşılık gelen değeri siler.

size: O zaman kadar kayıt edilmiş anahtar - değer ikili sayısını verir.

```
Map<String,String> userMap = new HashMap<String, String>();
      userMap.put("email", "ahmet@example.com");
      userMap.put("name", "Ahmet Zan");
      userMap.put("address", "Istanbul 34000");
      userMap.put("mobile", "5322100000");
      System.out.println("Kullanici adresi " + userMap.get("address"))
```

```
for(Map.Entry<String, String> pairs : userMap.entrySet()) {
          System.out.println(pairs);
}
```

Set

List Interface'ine benzeyen Set, verilen verileri bir dizin (index) kullanmadan saklamaktadır. Aynı zamanda Set arayüzü aynı elemanı iki kere saklamaya izin vermemektedir.

add (Object o) : Dizi içerisine bir eleman ekler.

clear: Dizi içerisindeki bütün elemanları siler.

contains (Object o): Bir eleman dizi içinde mi kontrol eder.

remove (Object o): Bir elemanı siler.

size: O ana kadar kaç eleman eklendiğini döndürür.

```
Set<String> treeSet = new TreeSet<String>();
      treeSet.add("Osman");
      treeSet.add("Ayse");
      treeSet.add("Osman");
      for(String str : treeSet) {
         System.out.println(str);
  }
```

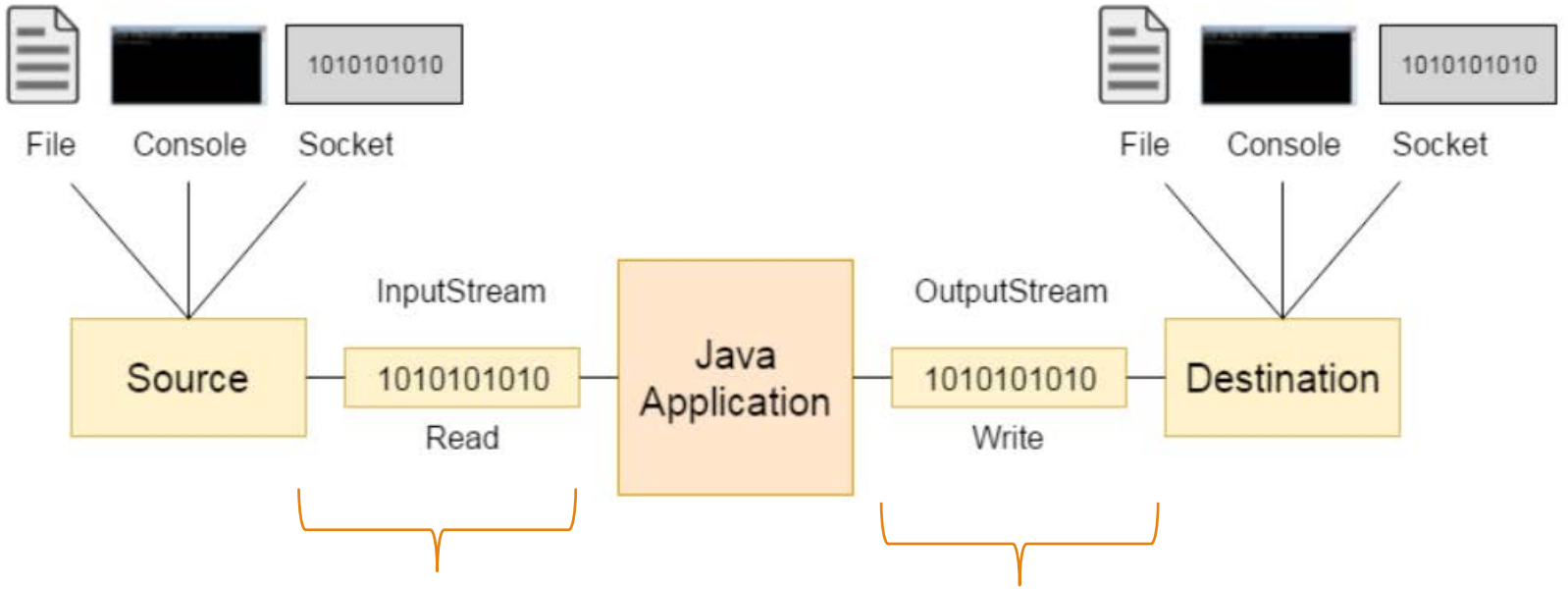
Nesne Tabanlı Programlama

I/O

DR. ŞAFAK KAYIKÇI

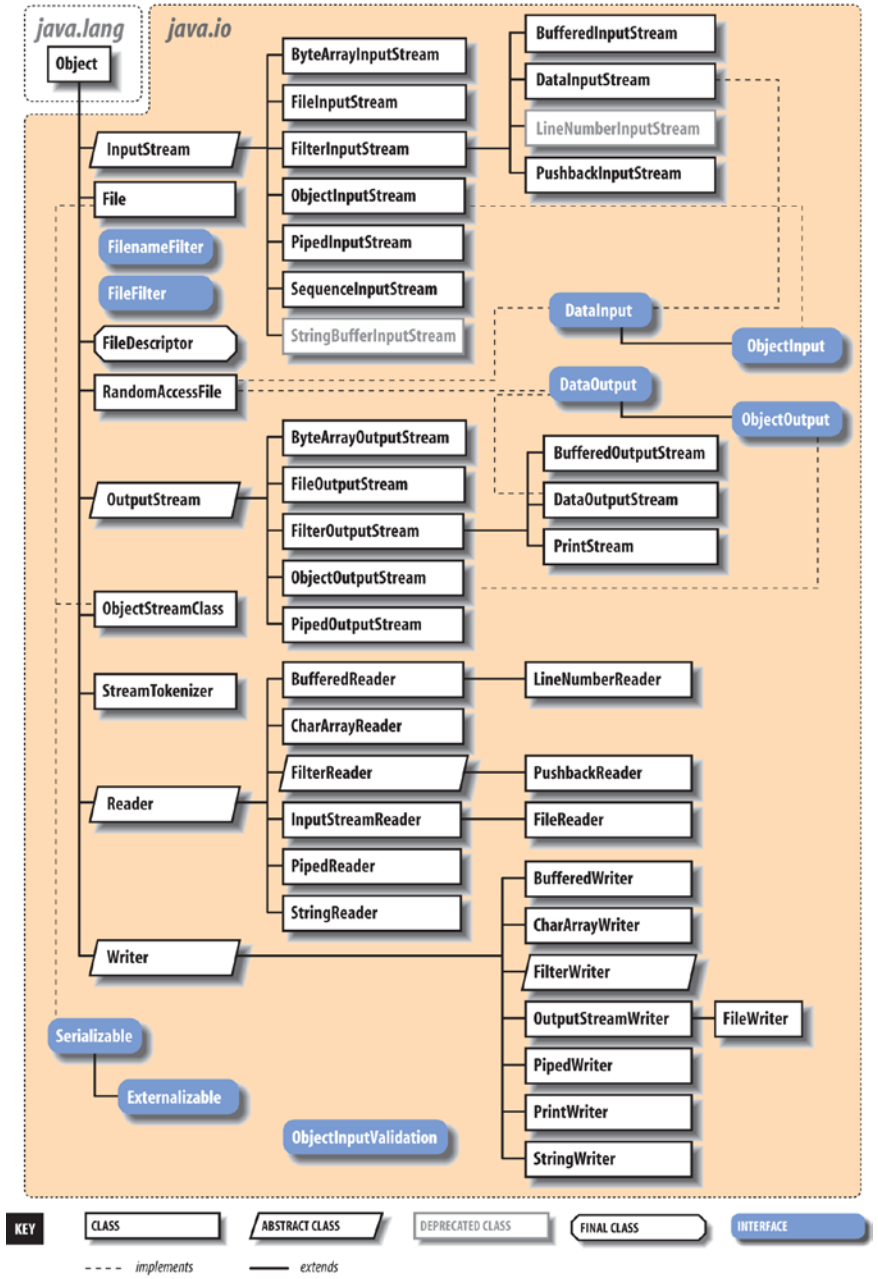
java IO (input – output)

Java'da , I/O işlemleri sıralanmış veri akışları (**Stream**) ile gerçekleştirilir. **java.io** paketi altında ele alınır.



InputStream : veri kaynağından okumak için

OutputStream: veri kaynağına yazmak için

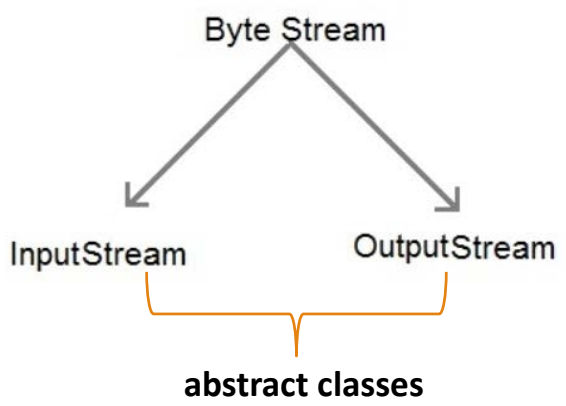

java.lang
java.io
Object
InputStream
File
FilenameFilter
FileFilter
FileDescriptor
RandomAccessFile
OutputStream
ObjectStreamClass
StreamTokenizer
Reader
Writer
Serializable
Externalizable
ObjectInputValidation
ByteArrayInputStream
FileInputStream
FilterInputStream
ObjectInputStream
PipedInputStream
SequenceInputStream
StringBufferInputStream
BufferedInputStream
DataInputStream
LineNumberInputStream
PushbackInputStream
DataInput
ObjectInput
DataOutput
ObjectOutput
ByteArrayOutputStream
FileOutputStream
FilterOutputStream
ObjectOutputStream
PipedOutputStream
BufferedOutputStream
DataOutputStream
PrintStream
BufferedReader
CharArrayReader
FilterReader
InputStreamReader
PipedReader
StringReader
LineNumberReader
PushbackReader
FileReader
BufferedWriter
CharArrayWriter
FilterWriter
OutputStreamWriter
PipedWriter
PrintWriter
StringWriter
FileWriter
KEY
CLASS
ABSTRACT CLASS
DEPRECATED CLASS
FINAL CLASS
INTERFACE
implements
extends

Akış Tipleri

1. **Byte Streams :** 1 byte (8bit) işlemler yapmak için kullanılır. FileInputStream , FileOutputStream en fazla kullanılandır.
2. **Character Streams :** 2 byte(16bit) Unicode şekilde okuma ve yazma yapar. FileReader, FileWriter en fazla kullanılandır. Buffered Streams : Satır satır okuma ve yazma yapar. Satır sonuna geldiğini EOL karakterleri ile anlar. ("\r\n") , ("\r") , ("\n"). BufferedReader , PrintWriter en fazla kullanılandır

*Character ve Byte streamlar arası Unicode'a göre (InputStreamReader ve OutputStreamWriter) dönüşümler yapılabilir.

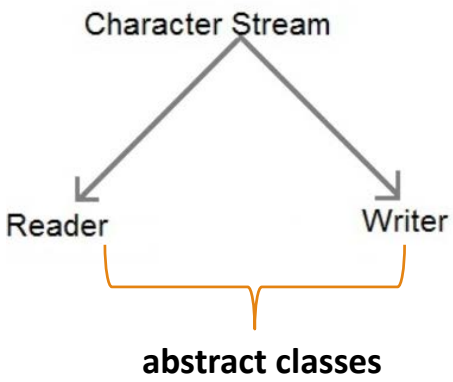
Byte Stream



En önemli iki method:
read() : verinin baytını okur.
write() : Verinin baytını yazar.

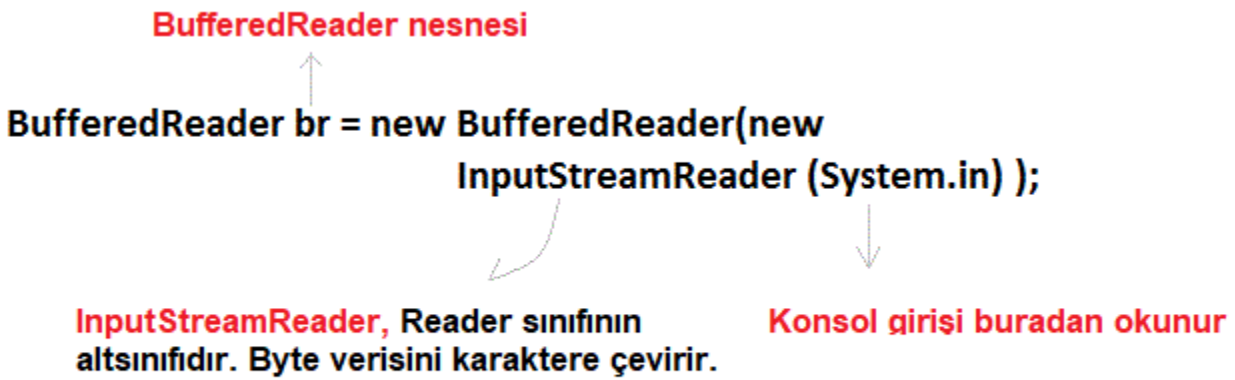
Stream class	Açıklama
BufferedInputStream	Tamponlu akış girişi için kullanılır.
BufferedOutputStream	Tamponlanmış akış çıkışiiçin kullanılır.
DataInputStream	Java standart veri türünü okumak için yöntem içerir
DataOutputStream	Java standart veri türü yazmak k için yöntem içerir
FileInputStream	Bir dosyadan okuyan giriş akışı
FileOutputStream	Bir dosyaya yazan çıktı akışı.
InputStream	Akış girişini tanımlayan soyut sınıf.
OutputStream	Akış çıktısını tanımlayan soyut sınıf.
PrintStream	print() ve println() yöntemlerini içeren çıkış akışı

Karakter Stream



Stream class	Açıklama
BufferedReader	Arabelleğe alınmış giriş akışını işler.
BufferedWriter	Arabelleğe alınmış çıktı akışını işler.
FileReader	Dosyadan okuyan giriş akışı.
FileWriter	Dosyaya yazan çıktı akışı.
InputStreamReader	Baytı karaktere çeviren giriş akışı
OutputStreamReader	Karakteri bayta çeviren çıktı akışı.
PrintWriter	print() ve println() yöntemlerini içeren çıkış akışı
Reader	Karakter akışı girişini tanımlayan soyut sınıf
Writer	Karakter akışı çıktısını tanımlayan soyut sınıf

Konsoldan okuma



Nesne Tabanlı Programlama

SERİALİZATION (SERİLEŞTİRME)

DR. ŞAFAK KAYIKÇI

serialization - deserialization

Serileştirme (**serialization**), bir nesneyi diskte/veritabanında kalıcı halde olabilen veya akış (stream) halinde gönderebilen bir byte dizisine dönüştürme işlemidir.

Bir sınıf , nesnesini başarıyla serileştirmek için **java.io** paketinde bulunan **Serializable** arayüzünü uygulamalıdır.

Serileştirme ve ters serileştirme (**deserialization**) işlemleri için **ObjectOutputStream** ve **ObjectInputStream** sınıfları kullanılır.

serialization - deserialization

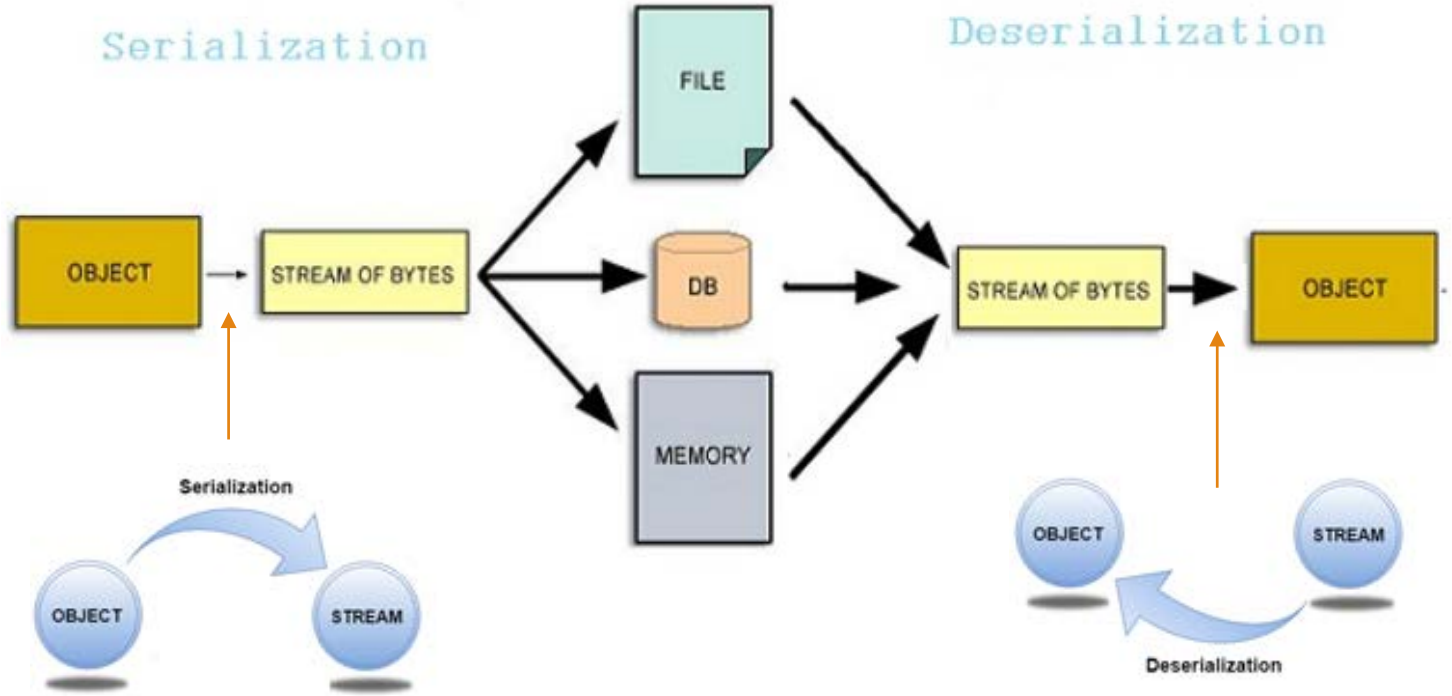
ObjectOutputStream : Dosyaya nesnenin durumlarını yazmak için kullanılır. Serileştirmeyi gerçekleştirmek için çeşitli yöntemler sağlar. writeObject() yöntemi kullanılır.

```
public final void writeObject(object x) throws IOException
```

ObjectInputStream : Daha önce serileştirilmiş olan nesneleri ve ilkel verilerine ters serileştirme (deserialization) işlemi yapar.

```
public final Object readObject() throws IOException,ClassNotFoundException
```

serialization - deserialization



serialization - deserialization

- Platform bağımsız
- Eğer bir sınıf serializable ise, kalıtım ile tüm alt sınıflara aktarılmış olur
- Aggregation (münesabet), Compostion (Oluşum) varsa hepsinde serializable olması gerekir
- Static değişkenler serileştirelemezler (Sınıfa ait olup, nesneye ait değillerdir)
- Bir sınıfın, bir alanını serileştirmek istemiyorsak *transient* kullanırız

Nesne Tabanlı Programlama

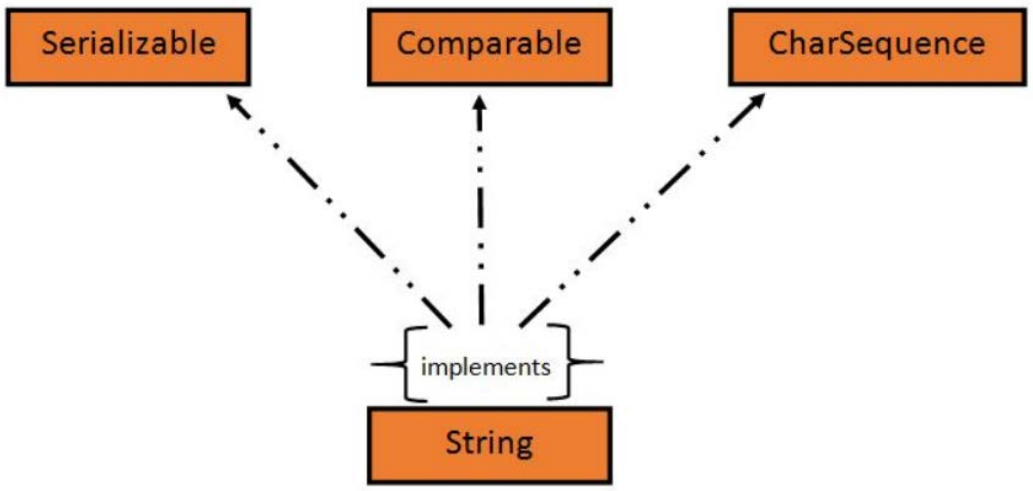
STRING (KATAR)

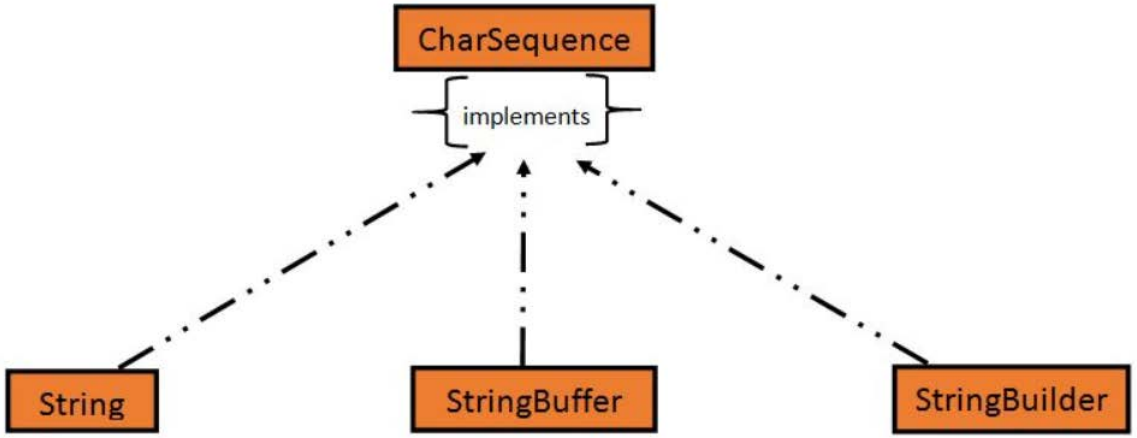
DR. ŞAFAK KAYIKÇI

String

String, karakter dizisini temsil eden bir nesnedir . **java.lang** paketinin içinde bulunan String sınıfı ile temsil edilir.

Serializable, Comparable ve CharSequence arayüzlerini uygular.





String oluşturma

1 - String literal

```
String s1 = "Hello Java";
```

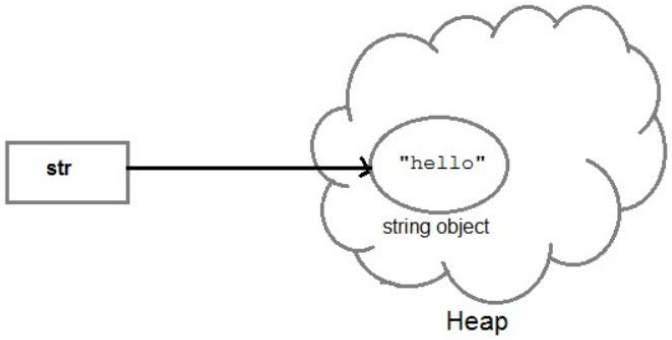
Bir String literal oluşturduğumuzda JVM önce string havuzunu kontrol (string pool) eder. String literal değeri havuzda zaten varsa, havuzdaki örneğe(instance) bir referans değişken döndürülür. Havuzda string yoksa, yeni bir string oluşturulup havuza yerleştirilir. String nesneleri , heap bellek içindeki dizge sabit havuzu **(string constant pool)** olarak bilinen özel bir bellek alanında depolanır .

2- new Keyword

```
String s1 = new String("Hello Java");
```

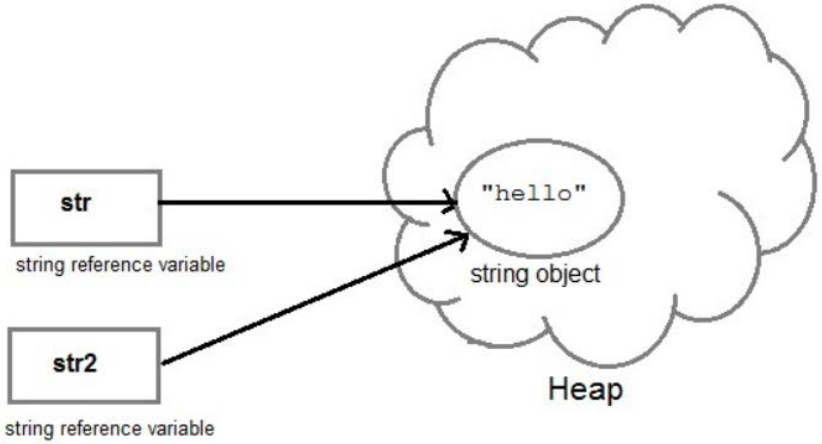
String

```
String str= "Hello";
```



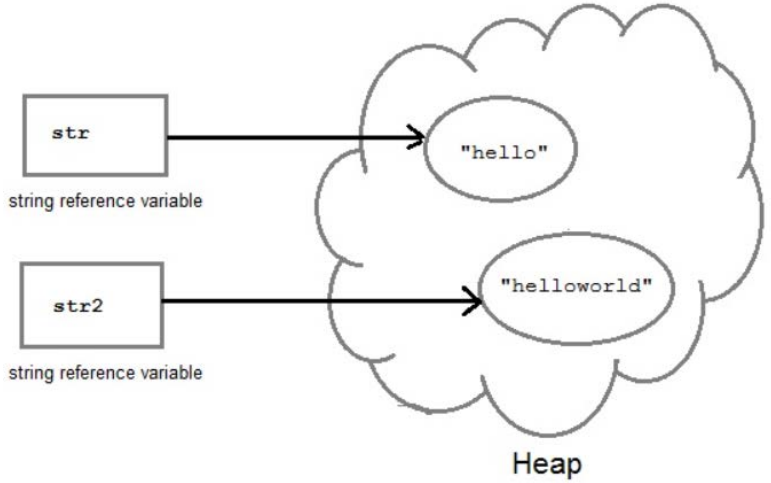
1-havuzda yoksa yeni yaratılır

```
String str2 = str;
```



2-havuzda varsa ona referans yaratılır

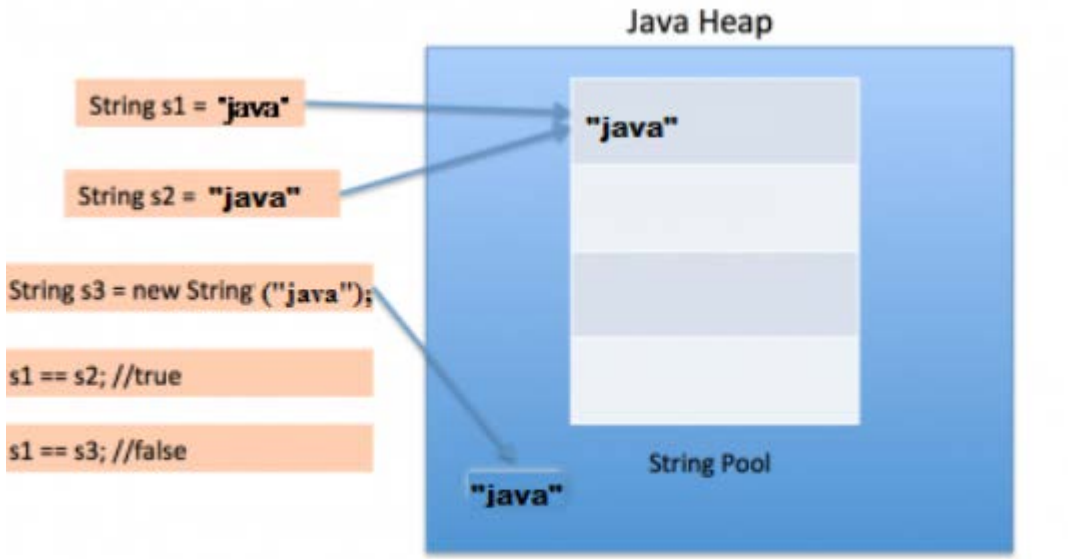
```
str2=str2.concat("world");
```



3-Ancak yeni stringi değiştirirsek, referansı değişir.

String Comparison

1) equals() yöntemi
2) == Operatörü

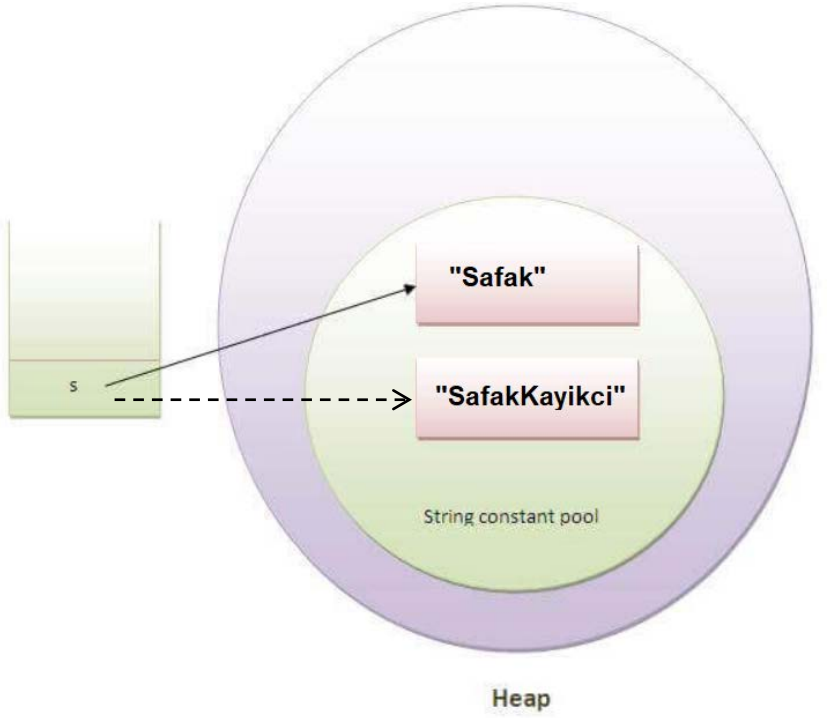


3) compareTo() yöntemi : Değerleri **alfabetik** olarak karşılaştırıp stringin diğerinden küçük, ona eşit veya büyük olup olmadığını gösteren bir **tamsayı** değeri döndürür.

Immutable

Oluşturulduktan sonra durumu değiştirilemeyen bir nesne değişmez (**immutable**) nesne olarak bilinir. String, Integer, Byte, Short, Float, Double ve diğer tüm sarmalayıcı sınıfları nesneleri değiştirilemez.

```
String s="Safak";
s.concat("Kayikci");
//concat() methodu  ucuna ekler. Ama yeni bir nesnede oluşturur.
System.out.println(s);
//sadece Safak yazdiracaktir. (s değişmemiştir (immutable))

s=s.concat ("Kayikci");
// s bu sefer oluşturulan yeni nesneyi göstermektedir.
System.out.println(s);
//Safak Kayikci yazacaktir
```



Sık kullanılan String methodları

length: string boyutu (karakter sayısı)

toLowerCase – toUpperCase: tamamını küçük harf – büyük harfe çevirme

replace: içerisindeki karakter yada karakter gruplarının değiştirme

trim: başındaki ya da sonundaki yer alan boşlukları ve tabları slime

substring: Stringleri parçalar. İki farklı kullanımı vardır; tek parametre verildiğinde o sayıdan sonraki karakter kümesini alır; iki parametre verildiğinde ise ilk parametreden başlayarak ikinci parametre sayısınca karakter kümesini alır.

contains: string içerisinde arama işlemi yapar

equals: İki stringin aynı olduğunu kontrol etmek kontrol etmek için kullanılır

concat: İki stringi birleştirmeyi sağlar. “+” operatörü de kullanılabilir.

Nesne Tabanlı Programlama

MULTITHREADING

DR. ŞAFAK KAYIKÇI

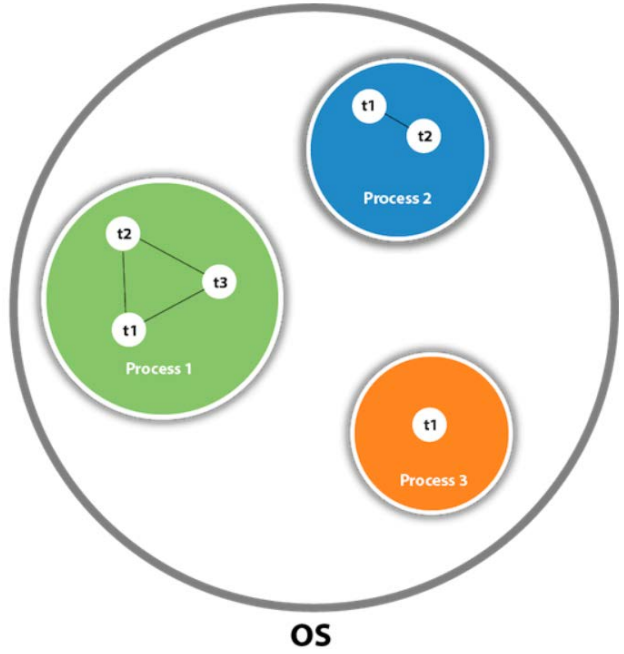
Thread (iplik)

Tek bir program akışı içerisinde birden fazla işlemin eşzamanlı olarak gerçekleştirilmesini sağlayan yapılardır.

Threadler (iplikler), Processler (işlemler) içerisinde bulunurlar. Bir process **enaz bir** tane thread barındırır.

Bir processe bağı bütün threadler aynı ortak hafıza alanını kullanırlar.

Threadlere light-weight processler denebilir.



Multitasking – Multithreading – Multiprocessing - Parallel Processing

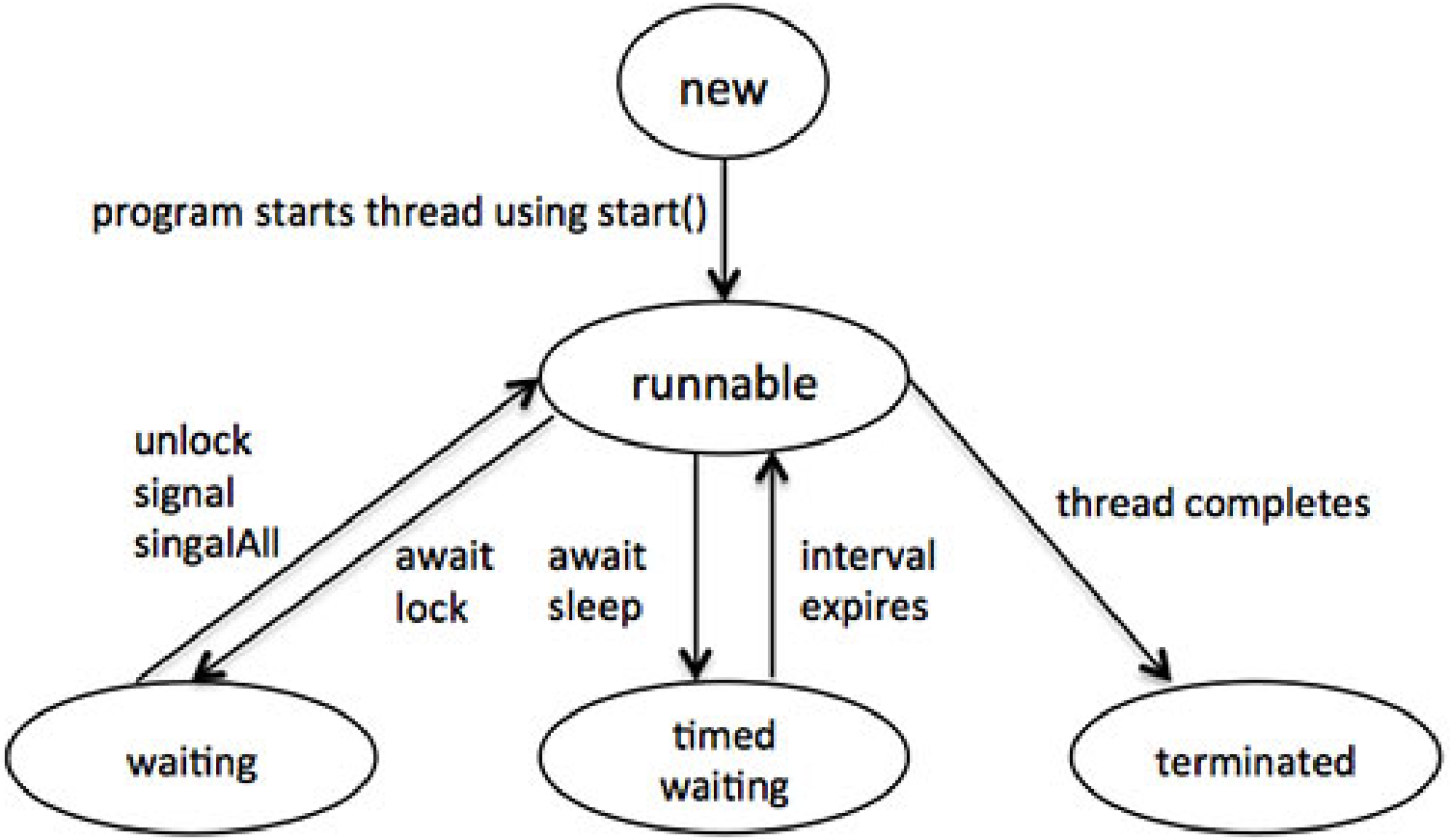
Multitasking (Çoklu görev) : Aynı anda birden fazla görevi yürütme işlemidir.

Multithreading (Çoklu iş parçacığı) : Aynı anda birden fazla iş parçacığı (thread) yürütme işlemidir. Multithreading aynı zamanda Thread-based Multitasking olarak da bilinir.

Multiprocessing (Çoklu işlem) : Çoklu görev (Multitasking) ile aynıdır, ancak çoklu işlemede birden fazla CPU söz konusudur.

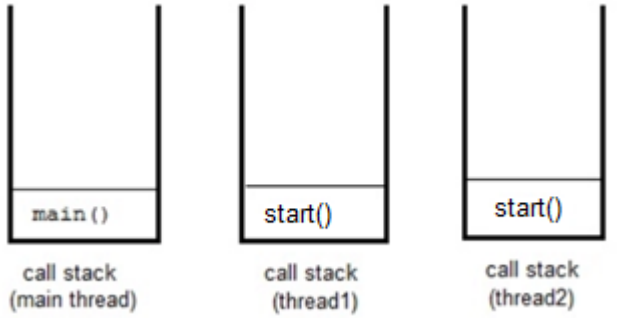
Parallel Processing (Paralel İşlem) : Tek bir bilgisayar sisteminde birden fazla CPU'nun faydalı olarak kullanılmasını ifade eder.

Thread Lifecycle



Main Thread

- Bir program çalışmaya başladığında JVM main() methodu içerisindeki kodu yürütmek için bir iş parçacığı (thread) oluşturur. Ana iş parçacığı (main thread) otomatik olarak oluşturulsa da, currentThread () yöntemini çağırılarak ona referans gönderilerek kontrol edilebilir.
- Diğer ipliklerin (thread) üretileceği ana ipliktir.
- Programı bitirmek için sonlandırılacak son iş parçacığıdır.




```
public class MainThread {
        public static void main(String[] args) {
                Thread t = Thread.currentThread();
                System.out.println("Name of thread is " + t);
        }
}
```

2 Şekilde üretilir

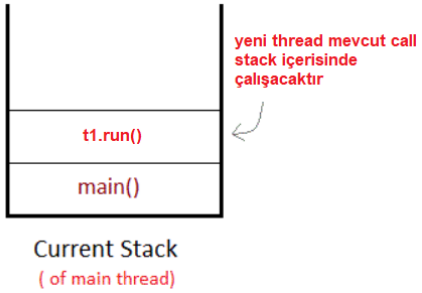
1) Thread Sınıfını extend ederek

```
public class MyThread extends Thread {
	public void run() {
		System.out.println("thread calisiyor...");
	}

	public static void main(String args[]) {
		MyThread t1 = new MyThread();
		t1.start();
		// iki defa start ??
	}
}
```

2)Runnable arayüzünü uygulayarak

```
public class MyThread2 implements Runnable{
	public void run() {
		System.out.println("thread calisiyor...");
	}

	public static void main(String args[]) {
		MyThread2 m1 = new MyThread2();
		Thread t1 = new Thread(m1);
		t1.start();
	}
}
```

start() yerine run() çağırılırsa iş parçacığına yeni bir call stack (çağrı yığını) tahsis edilmeyecek ve main iş parçacığının call stack içerisinde çalışmaya başlayacaktır. Dolayısıyla Multithreading olmayacaktır.



Methodlar

getName(): thread ismini almak için kullanılır

getID() : thread ID'sini almak için kullanılır

getPriority(): thread öncelik sırasının almak için kullanılır

isAlive(): thread'in çalışır durumda olup olmadığını bildirir

join(): thread'in yok olmasını bekler

run(): thread için başlangıç noktasıdır

sleep(): belli bir süre için aslıya alır

start(): thread'in run metodunu çağırarak başlatır

CurrentThread(): o an çalışan iş parçacığını getirir

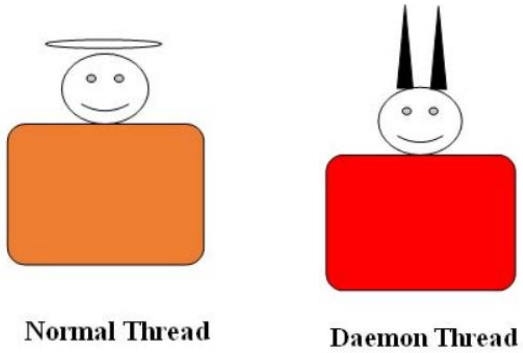
Deamon Thread

Daemon iş parçacığı, kullanıcı iş parçacıklarına destek sağlayan düşük öncelikli bir iş parçacığıdır.

Bu iş parçacıkları kullanıcı tanımlı ve sistem tanımlı olabilir.

Garbage Collection thread (Çöp toplama iş parçacığı), arka planda çalışan, sistem tarafından üretilen daemon iş parçacığından biridir.

Daemon iş parçacığı, tüm iş parçacıkları yürütmeyi bitirene kadar JVM'nin var olmasına izin verir. Bir JVM, arka plan programı evrelerini bulduğunda, iş parçacığını sonlandırır ve sonra kendini kapatır, Daemon iş parçacığının çalışıp çalışmadığını umursamaz.

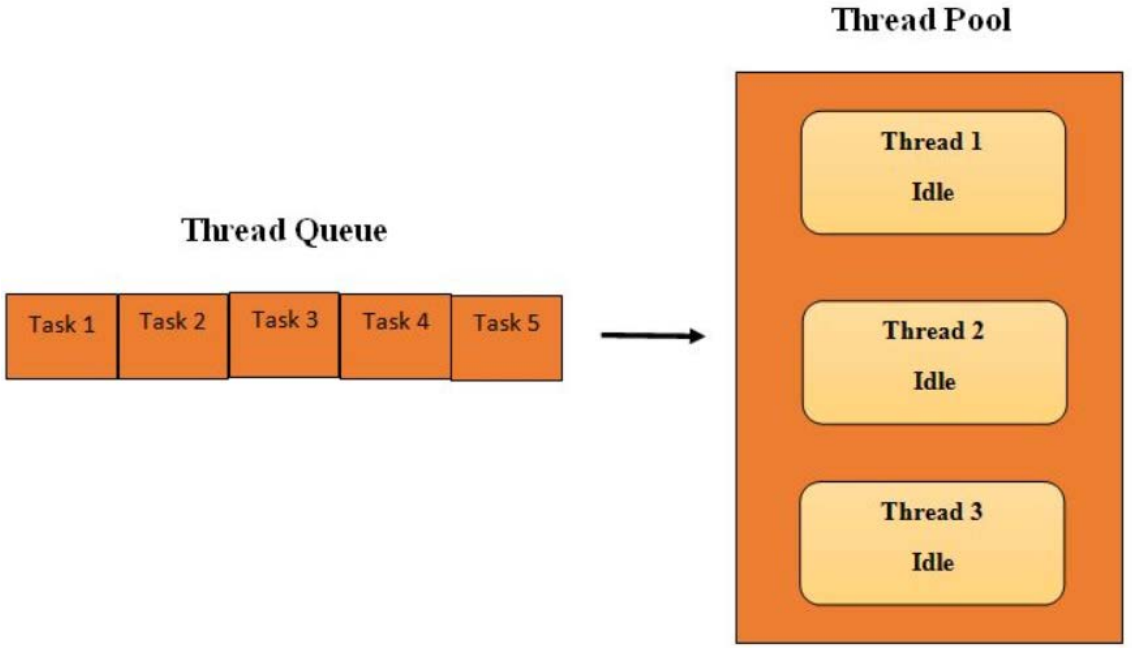


public final void setDaemon(boolean on)

public final booleanisDaemon()

Thread Pool

Java'da, mevcut görevi yürütmek için önceden oluşturulmuş iş parçacıkları yeniden kullanmak için kullanılır. Ayrıca, iş parçacığı döngüsünde veya kaynak atma işleminde herhangi bir sorun ortaya çıkarsa çözüm sağlar. Java iş parçacığı havuzunda (thread pool) bir iş parçacığı grubu oluşturulur, bir iş parçacığı seçilir ve iş atanır ve iş tamamlandıktan sonra gruba geri gönderilir.



Thread Priority

İş parçacığı öncelikleri, bir iş parçacığının diğerlerine göre nasıl ele alınacağına karar veren 1 ile 10 arasında tam sayılardır. Öncelik, JVM veya programcının kendisi tarafından verilir.

public static int MIN_PRIORITY →1

public static int NORM_PRIORITY →5 (default)

public static int MAX_PRIORITY →10

NOT : Öncelik değeri, daha yüksek öncelikli bir iş parçacığının her zaman düşük öncelikli iş parçacığından önce yürütüleceğini garanti edemez. Yürütme için iş parçacığı seçimi, platforma bağlı olan iş parçacığı zamanlayıcısına (thread scheduler) bağlıdır.

Nesne Tabanlı Programlama

SWING

DR. ŞAFAK KAYIKÇI

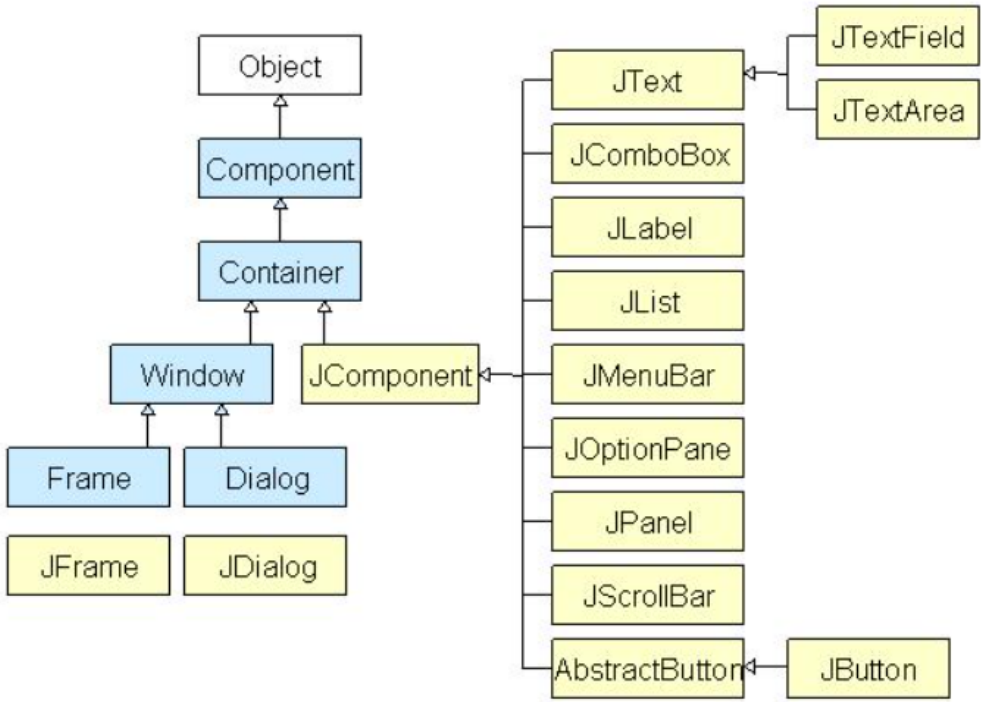
Swing

Swing, Grafik Kullanıcı Arayüzü (Graphical User Interface -GUI) bileşenlerini içeren bir araç setidir. GUI, Java uygulamaları için kullanımı kolay bir görsel deneyim oluşturur. Esas olarak, kullanıcının bir uygulamayla etkileşime girebileceği düğmeler, etiketler, pencereler vb. gibi grafik bileşenlerden oluşur.

Java programları için GUI sağlayan bir API olan Java Foundation Classes'ın (JFC) bir parçasıdır.

Java Swing kütüphanesi, daha eski, platforma bağlı bir GUI araç seti olan Java Abstract Widget Toolkit (AWT) üzerine inşa edilmiştir.

Java Swing Class Hierarchy Diagram



Java Swing'deki tüm bileşenler, konteyner sınıflarına eklenebilecek JComponent'tir

Container (Kab)

Konteyner sınıfları, üzerinde başka bileşenlere sahip olabilen sınıflardır. BirJava GUI oluşturmak için en az bir konteyner nesnesine ihtiyaç vardır.

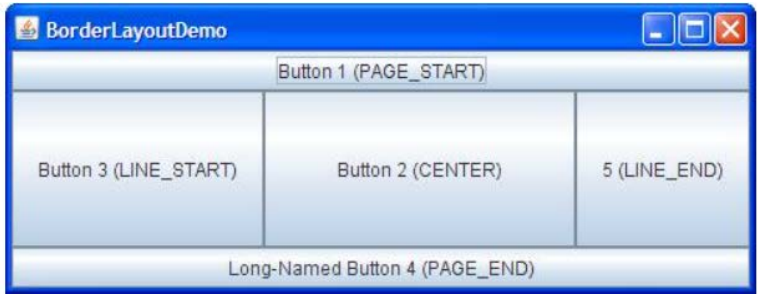
3 tür Java Swing kabı vardır:

1. **Panel**: Saf bir kaptır ve kendi başına bir pencere değildir. Panelin görevi bileşenleri bir pencerede düzenlemektir.
2. **Frame**: Başlığı ve simgeleri içeren bir penceredir.
3. **Dialog** : Bir mesajın görüntülenmesi gerektiğinde açılan bir açılır pencere gibi düşünülebilir. Frame gibi tam olarak çalışan bir pencere değildir.

Layout Manager (Düzen Yöneticisi)

Layout Manager, bir konteyner içindeki GUI java bileşenlerini düzenlemek ve sırlamak için kullanılır.

BorderLayout (Sınır Düzeni)



BorderLayout bileşenleri beş alana kadar yerleştirir: top, bottom, left, right, center. JFrame için varsayılan (default)düzen yöneticisidir.

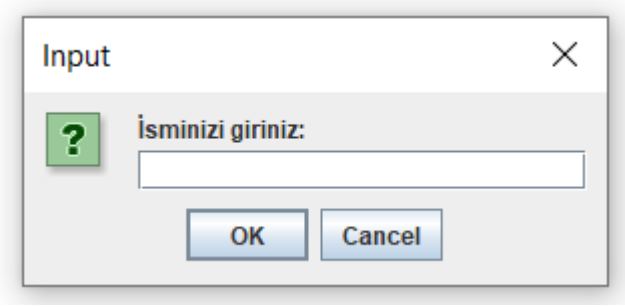
FlowLayout (Akış Düzeni)

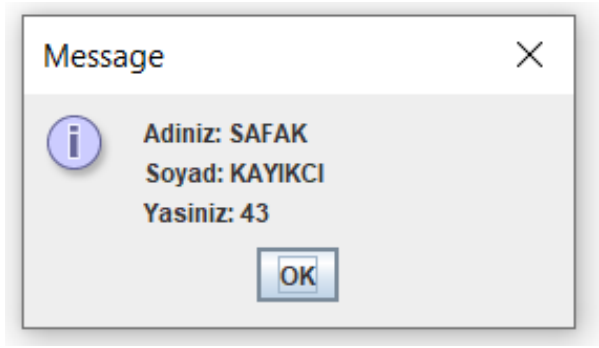
JPanel için varsayılan düzen yöneticisidir. Bileşenleri tek bir sıra halinde birbiri ardına yerleştirir.

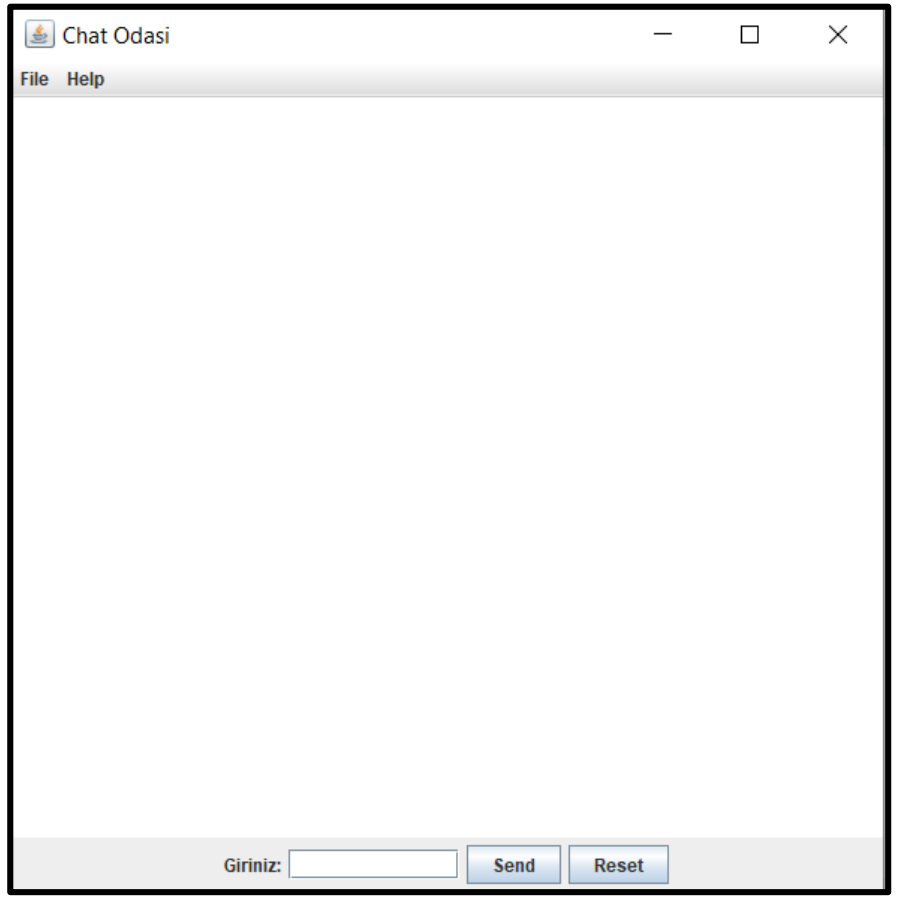
GridBagLayout

Bileşenleri bir hücre ızgarası içine yerleştirerek hizalar ve bileşenlerin birden fazla hücreye yayılmasına izin verir.

Örnekler


Input
İsminizi giriniz:
OK
Cancel


Message
Adiniz: SAFAK
Soyad: KAYIKCI
Yasiniz: 43
OK


Chat Odasi
File Help
Giriniz:
Send
Reset