

# Java Input/Output API

# java.io.File Sınıfı

- Hiyerarşik **path bilgisini** işletim sisteminden **bağımsız** biçimde ifade etmeyi sağlar
- **Bir dosyayı** ya da **bir dizini** ifade edebilir
- Eğer file nesnesi bir dizin ise, **list() metodu** ile altındaki dosyalara erişilebilir
- list() metodu **dosya isimlerinden** oluşan String array döner
- File sınıfındaki metotlar ile dosya, dizin **yaratma ve silme** işlemleri de yapılabilir
- Dosya ve dizinlerin okuma, yazma ve çalıştırma **hakları** değiştirilebilir

# java.io.File Sınıfı

```
File desktop = new File("/home/ksevindik/Desktop");  
if(desktop.isDirectory()) {  
    for(String s:desktop.list()) {  
        System.out.println(s);  
    }  
} else {  
    System.out.println(desktop.getPath());  
}
```

File bir dizin ise content'i listelenebilir

```
File newFile = new File("/home/ksevindik/Desktop/test.txt");  
newFile.createNewFile();
```

```
if(newFile.canWrite()) {  
    newFile.delete();  
}
```

Temp dosyanın JVM sonlandığında otomatik silinmesi için deleteOnExit gereklidir

```
File tempFile = File.createTempFile("prefix", "suffix");  
tempFile.deleteOnExit();
```

İşletim sisteminin temp dizininde yaratır

```
File.createTempFile("prefix", "suffix", folder);
```

Temp dosyayı belirtilen dizinde yaratır

# java.io.File Sınıfı

```
File file = new File("/home/ksevindik/Desktop/test.sh");
```

```
boolean exists = file.exists();
```

```
boolean hidden = file.isHidden();
```

```
boolean canExecute = file.canExecute();
```

```
boolean canRead = file.canRead();
```

```
boolean canWrite = file.canWrite();
```

```
file.setExecutable(false);
```

```
file.setReadable(false);
```

```
file.setWritable(false);
```

```
System.out.println(File.pathSeparator);
```

```
System.out.println(File.separator);
```

Dosyanın mevcut olup olmadığı, gizli dosya olup olmadığı, hakları test edilebilir ve değiştirilebilir

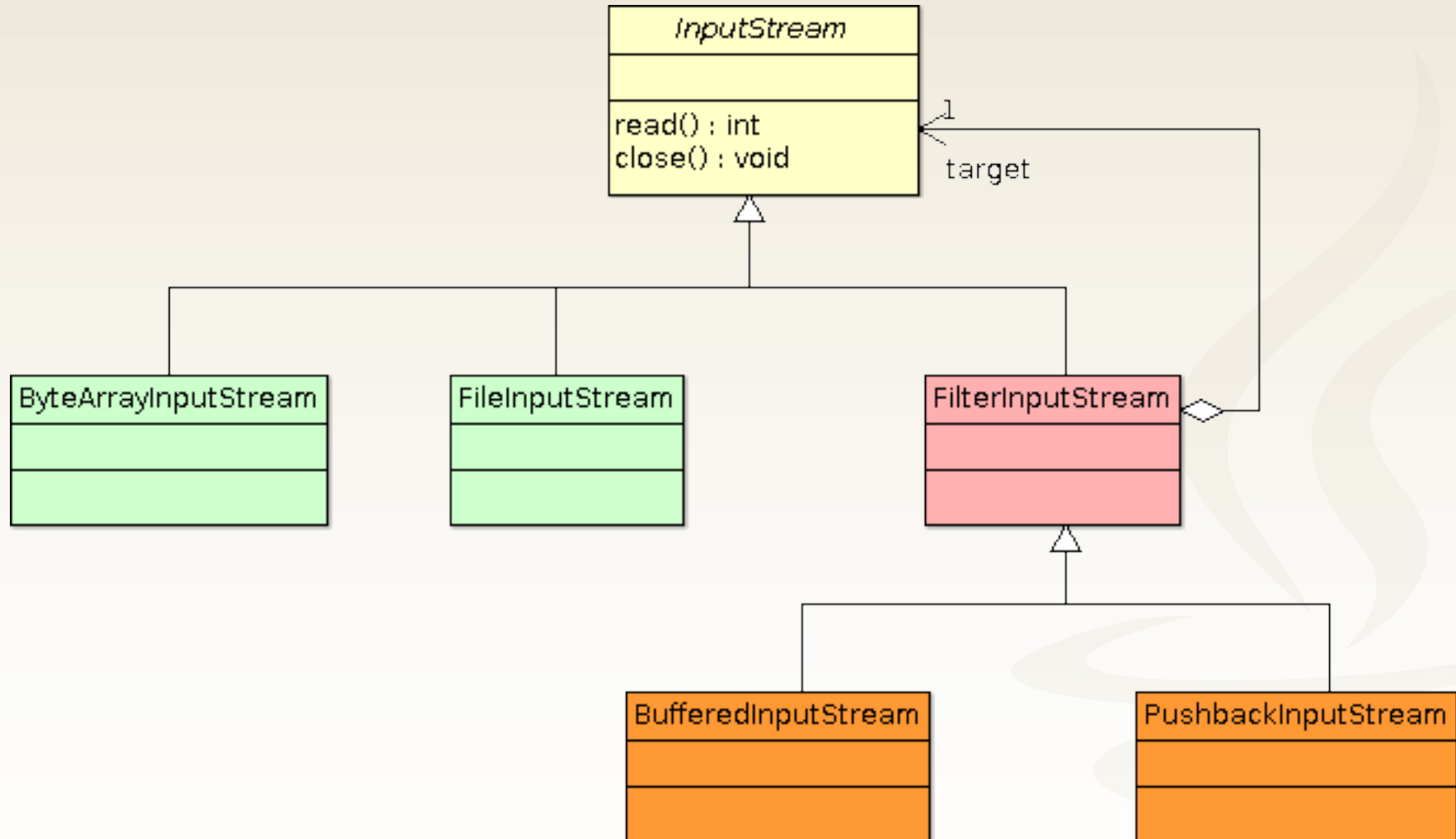
Unix için :  
Windows için ;

Unix için /  
Windows için \

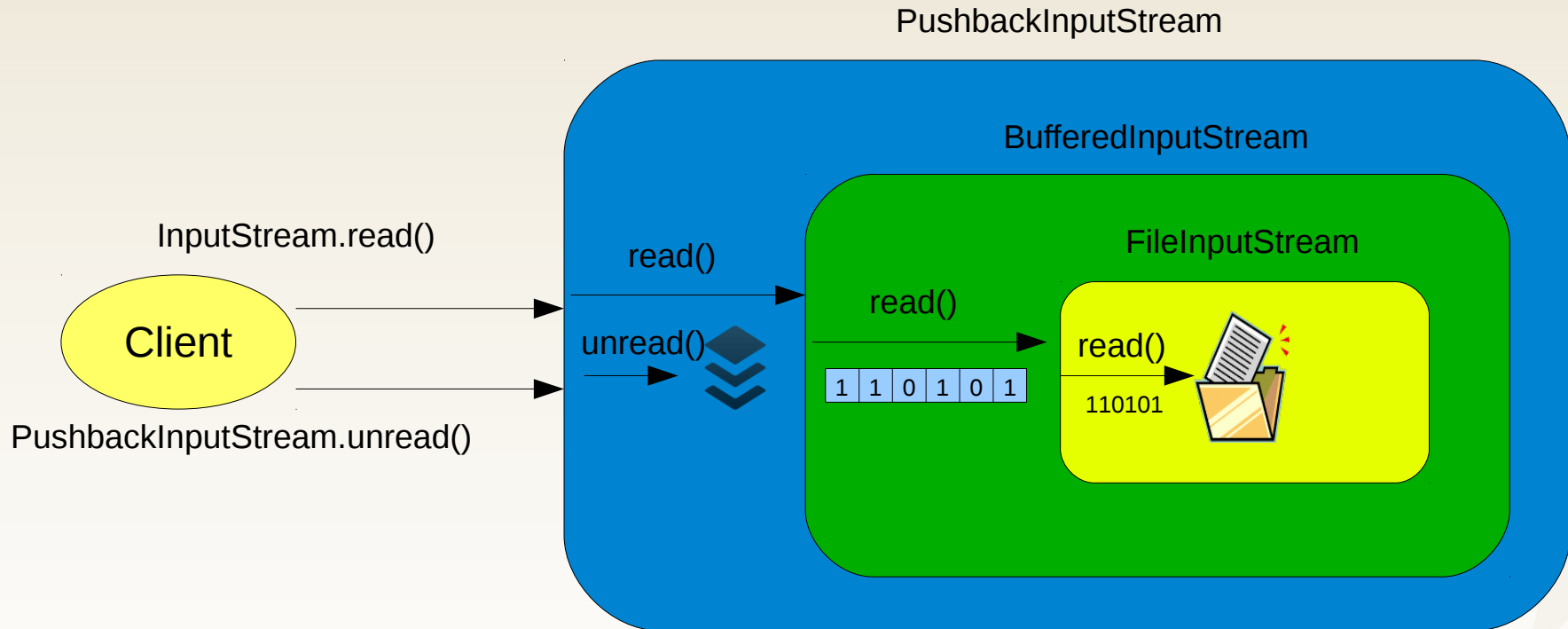
# Binary & Character Streams

- Java I/O API'si byte stream ile uğraşmak için **InputStream & OutputStream** arayüzlerini sunar
- Karakter stream ile uğraşmak için ise **Reader & Writer** arayüzlerini sunar
- İki stream türü arasında dönüşüm gerçekleştirmek için ise **InputStreamReader & OutputStreamWriter** sınıfları mevcuttur

# InputStream Mimarisi

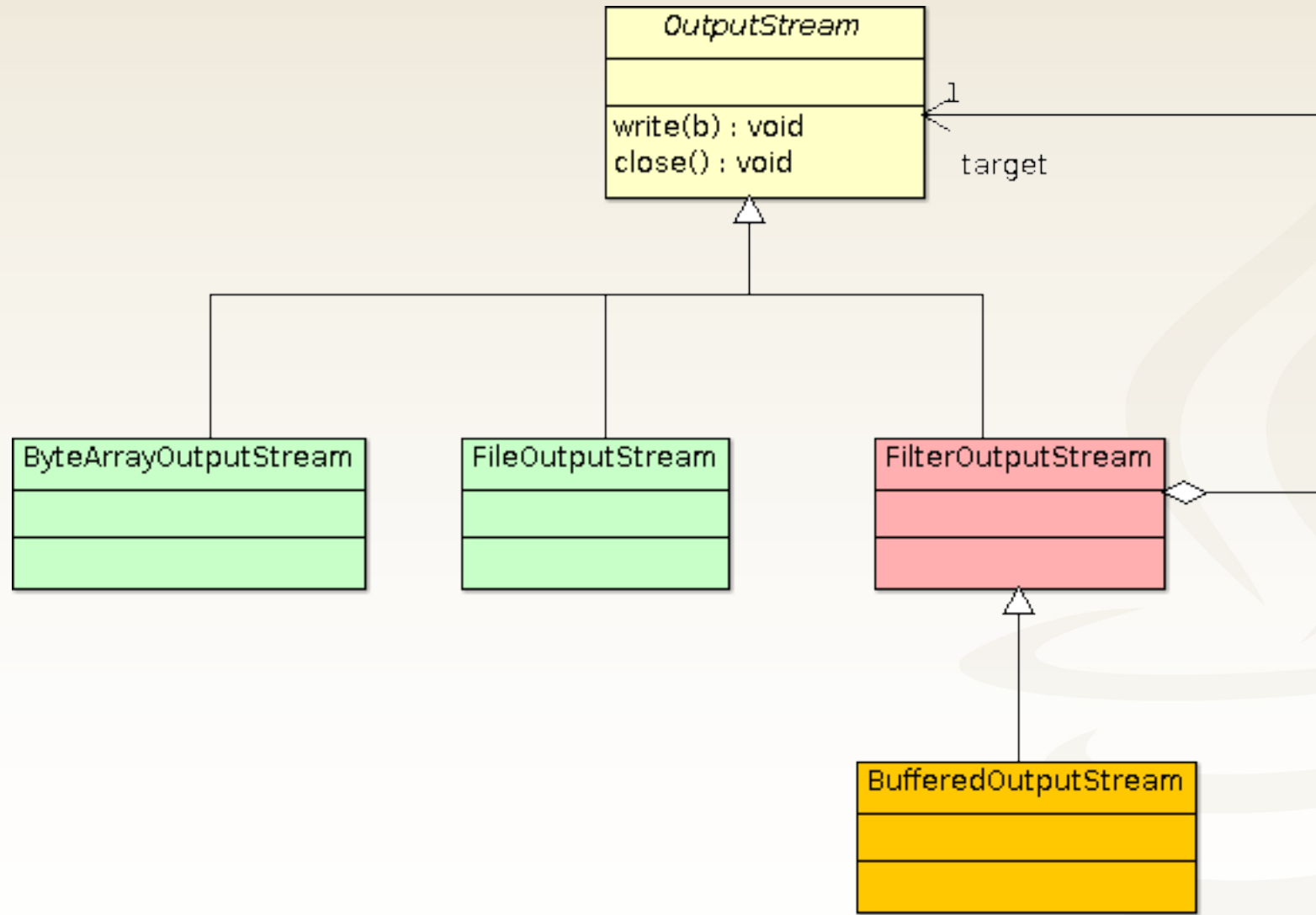


# InputStream Oluşturulması



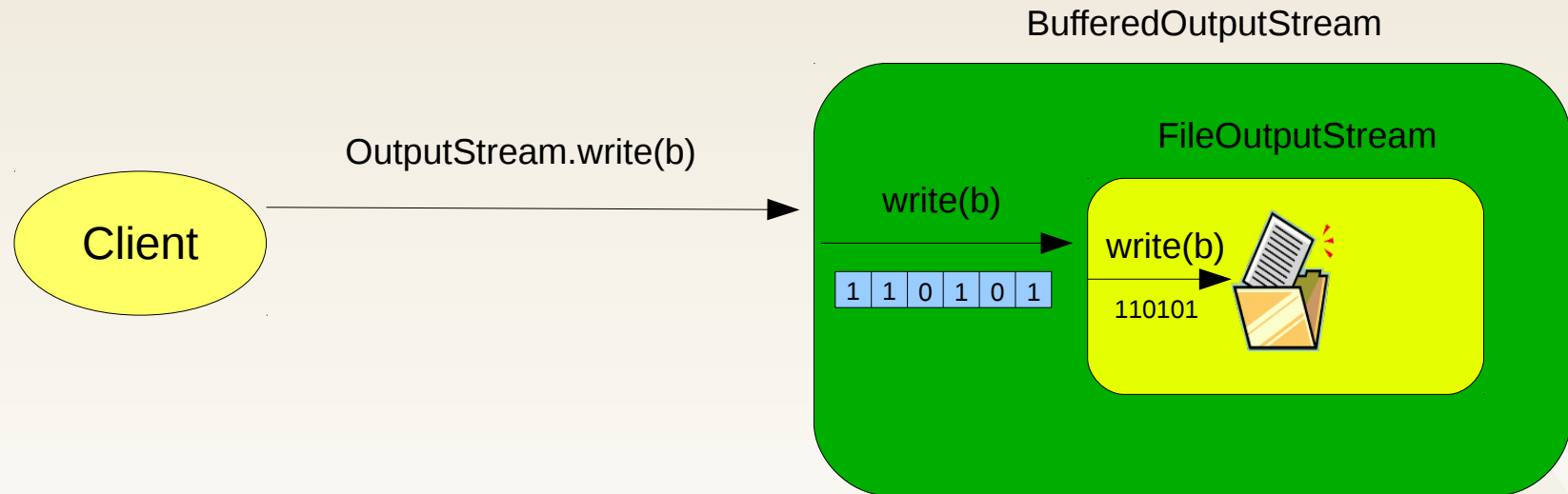
```
PushbackInputStream is = new PushbackInputStream(  
    new BufferedInputStream(  
        new FileInputStream ("/myfile.txt")));  
  
is.read();  
  
is.unread();  
  
is.close();
```

# OutputStream Mimarisi



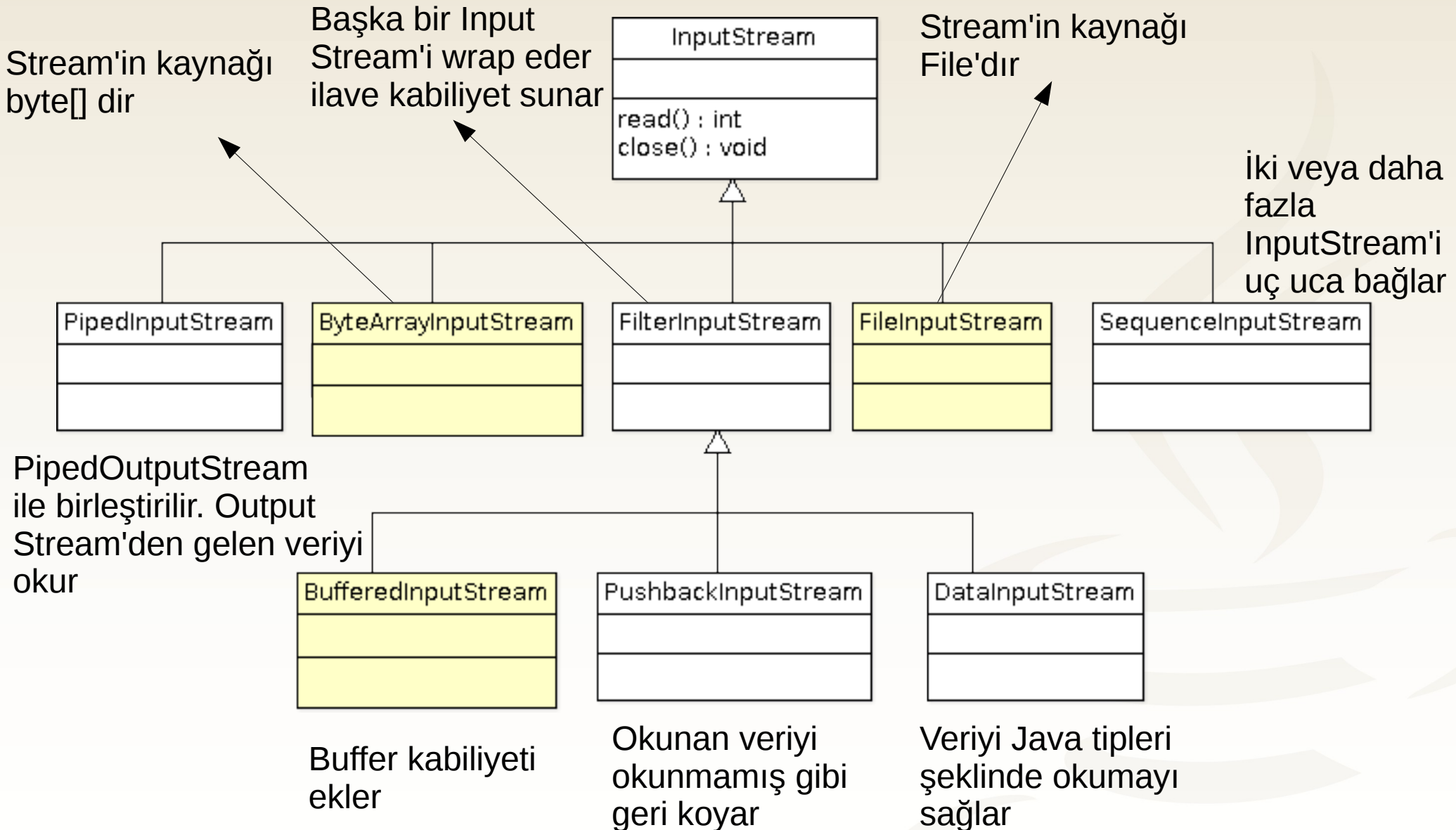


# OutputStream Oluşturulması

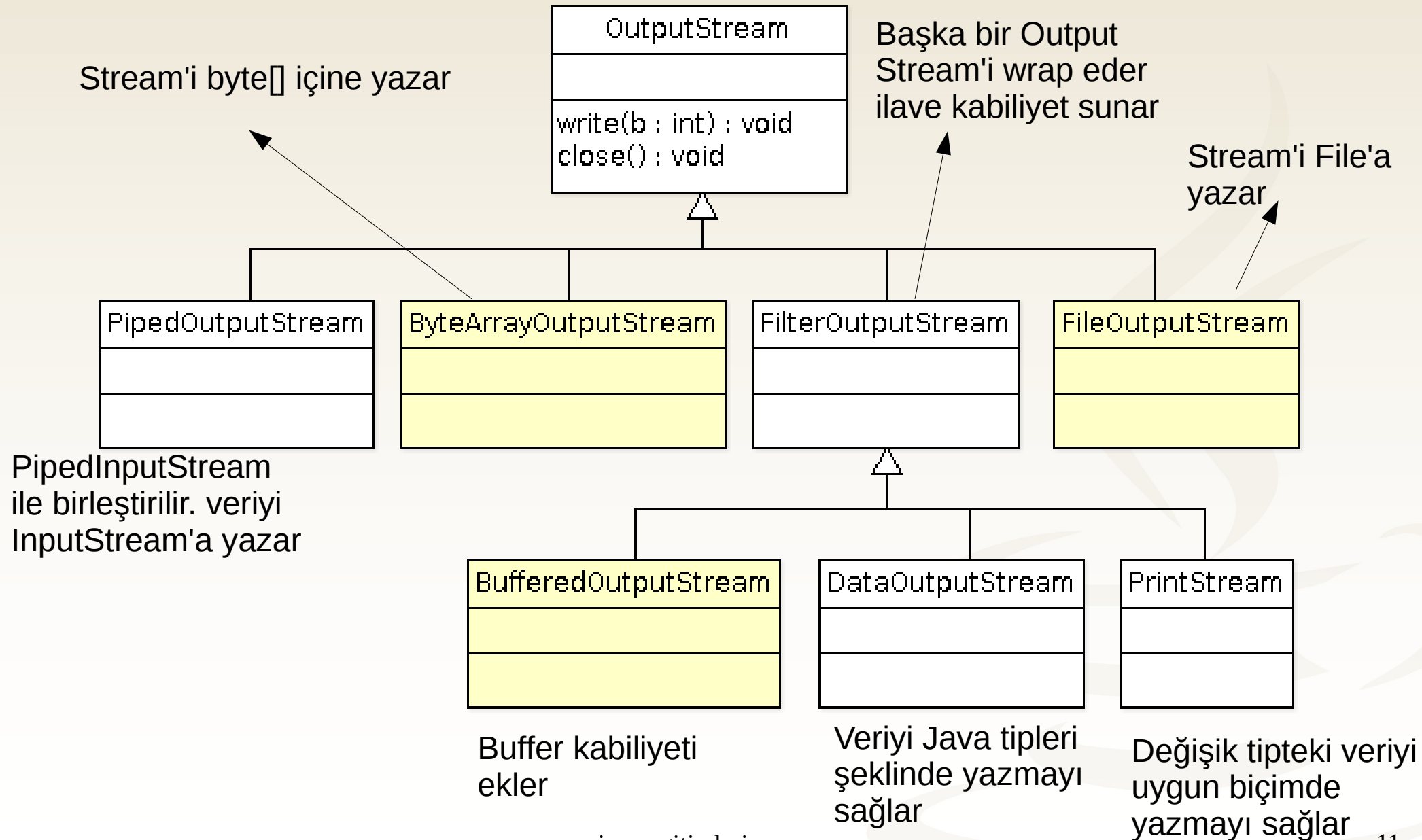


```
OutputStream os = new BufferedOutputStream(  
    new FileOutputStream ("/myfile.txt"));  
  
os.write(b);  
  
is.close();
```

# InputStream Hiyerarşisi

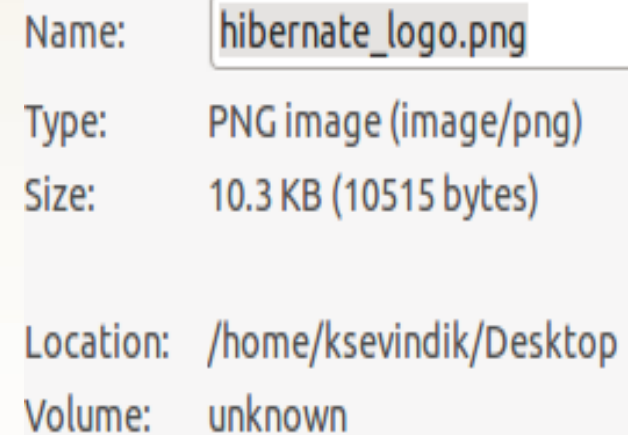


# OutputStream Hiyerarşisi



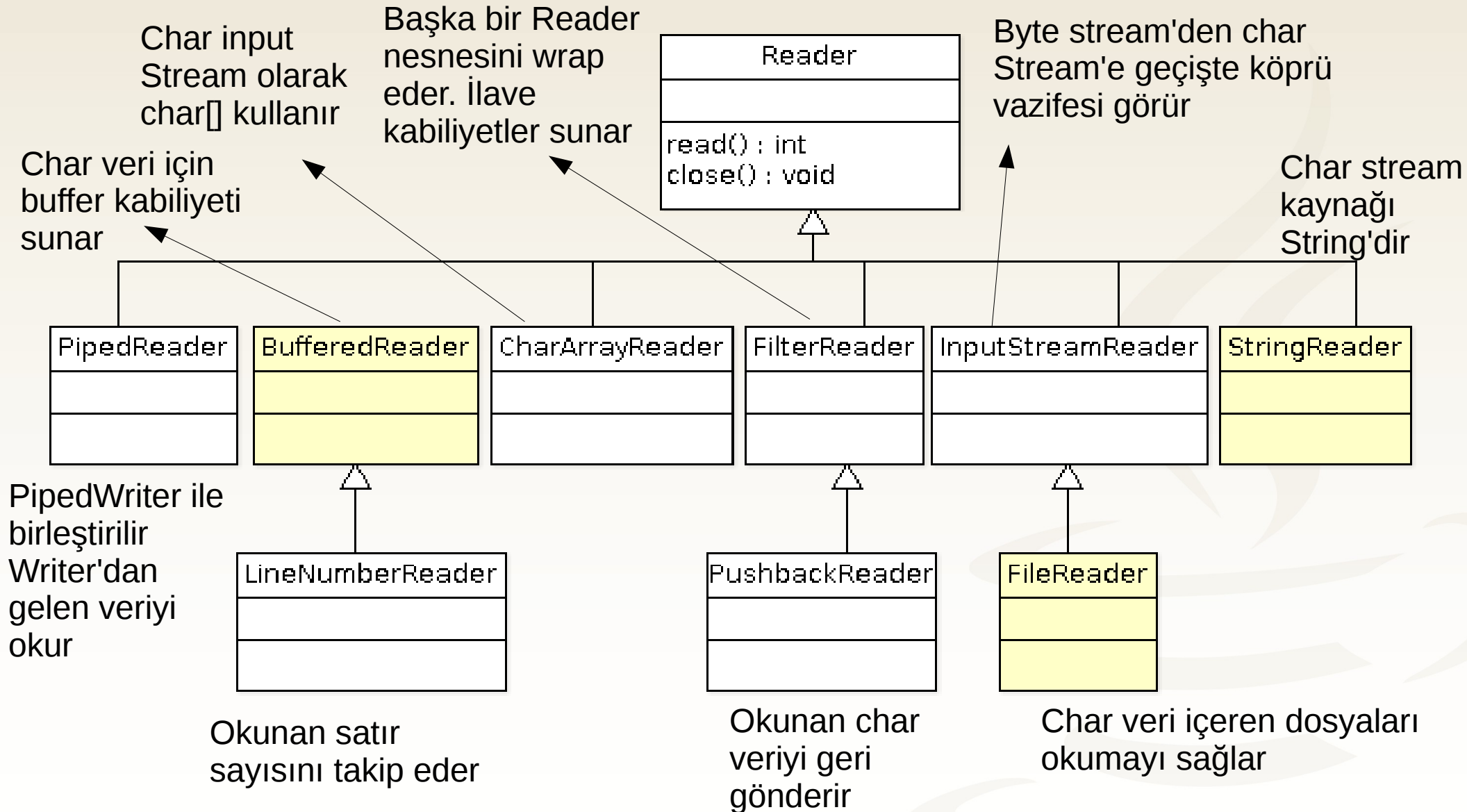
# InputStream & OutputStream Kullanım Örneği

```
File file = new File("/home/ksevindik/Desktop/hibernate_logo.png");  
FileInputStream fin = new FileInputStream(file);  
BufferedInputStream bin = new BufferedInputStream(fin);  
ByteArrayOutputStream bout = new ByteArrayOutputStream();  
  
while(bin.available()!=0) {  
    int i = bin.read();  
    bout.write(i);  
}  
  
bin.close();  
  
byte[] image = bout.toByteArray();  
  
System.out.println(image.length);  
  
10515
```

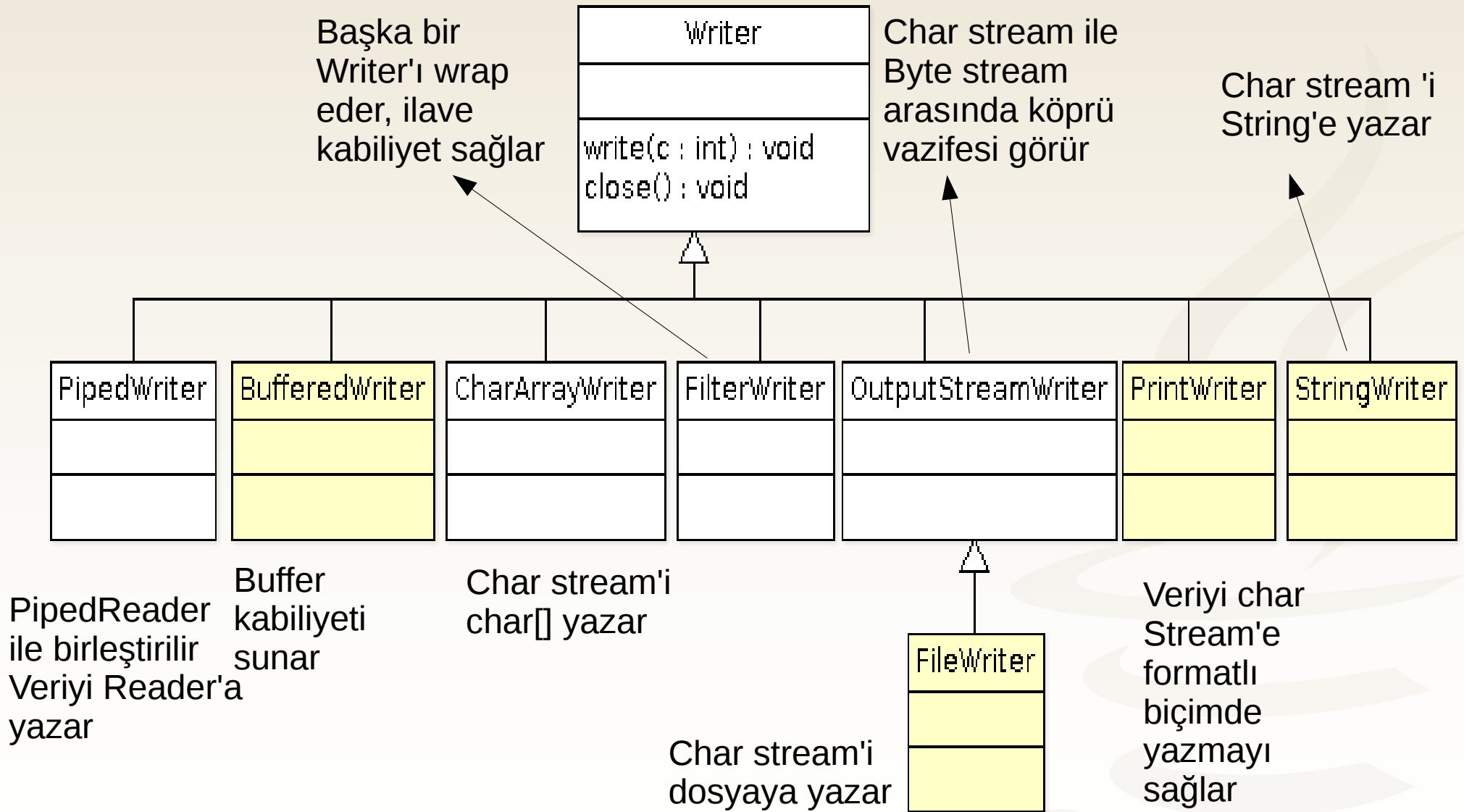


Name:	hibernate_logo.png
Type:	PNG image (image/png)
Size:	10.3 KB (10515 bytes)
Location:	/home/ksevindik/Desktop
Volume:	unknown

# Reader Hiyerarşisi



# Writer Hiyerarşisi



# Reader & Writer Kullanım Örneği

```
File file = new File("/home/ksevindik/Desktop/readme.txt");  
FileReader fr = new FileReader(file);
```

```
BufferedReader br = new BufferedReader(fr);
```

```
FileWriter fw = new  
FileWriter("/home/ksevindik/Desktop/readme2.txt", false);
```

```
while(br.ready()) {  
    int i = br.read();  
    fw.write(i);  
}
```

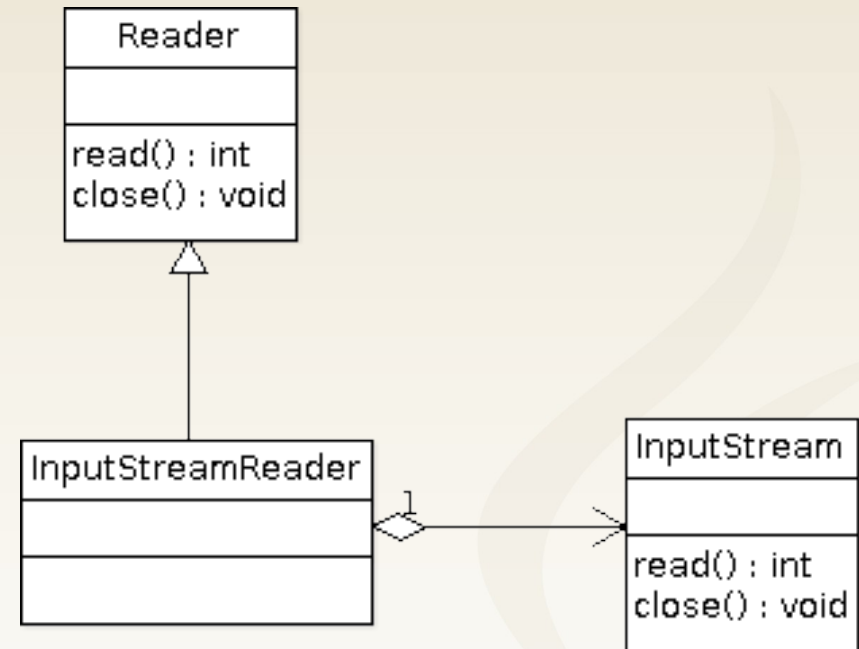
```
br.close();  
fw.close();
```

Name:	readme.txt
Type:	plain text document (text/
Size:	123 bytes (123 bytes)
Location:	/home/ksevindik/Desktop
Volume:	unknown

Name:	readme2.txt
Type:	plain text document (text/p
Size:	123 bytes (123 bytes)
Location:	/home/ksevindik/Desktop
Volume:	unknown

# InputStreamReader ile Binary - Character Stream Dönüşümü

- Byte stream'i karakter stream'e dönüştürmek için kullanılır
- Bunun için belirli bir **charset**'e ihtiyaç vardır



11001100010

Byte stream



AABFGHIIH

Character stream



# InputStreamReader Kullanımı

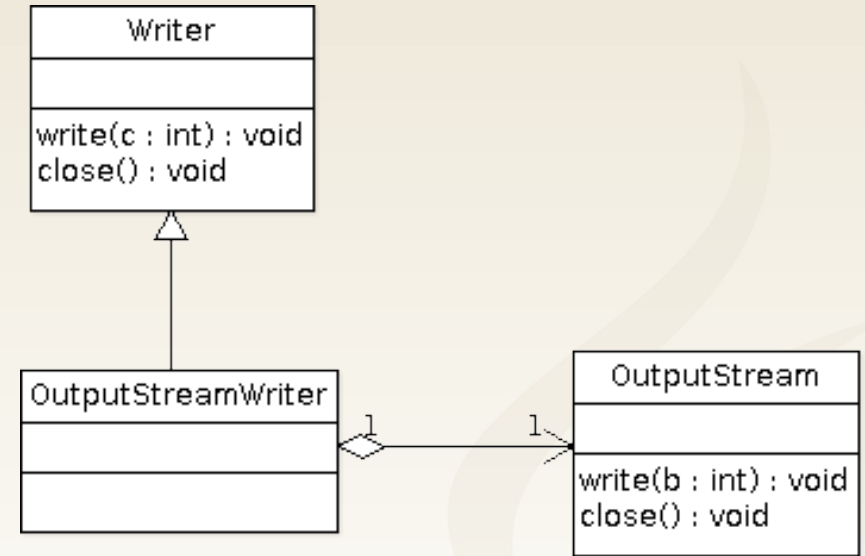
```
File file = new File("/home/ksevindik/Desktop/readme.txt");  
FileInputStream fin = new FileInputStream(file);  
InputStreamReader streamReader =  
    new InputStreamReader(fin, "utf-8");  
  
char c = (char)streamReader.read();
```

# OutputStreamWriter ile Character

JAVA Eğitimleri

## – Binary Stream Dönüşümü

- Karakter stream'i byte stream'e dönüştürmek için kullanılır
- Bu dönüşüm için de yine belirli bir **charset** kullanılır



AABFGHIHH

Character stream



11001100010

Byte stream

# OutputStreamWriter Kullanımı

```
File file = new File("/home/ksevindik/Desktop/readme2.txt");  
FileOutputStream fout = new FileOutputStream(file);  
OutputStreamWriter streamWriter =  
    new OutputStreamWriter(fout, "utf-8");  
streamWriter.write('A');
```

# Standart I/O Nesneleri

- **System.in, System.out ve System.err** nesneleri Java'daki standart I/O nesneleridir
- **System.in**, InputStream nesnesidir
- Sistemde standart input olarak tanımlı keyboard gibi bir **input aracından veri okumaya** yardımcı olur
- **System.out ve System.err PrintStream** nesneleridir
- Sistemde standard output olarak tanımlı **ekrana veri yazılmasını** sağlarlar

# Standart I/O Nesneleri

```
System.out.println("Hello World!!!");
```

```
System.err.println("Error!!!");
```

```
int i = System.in.read();
```

# Standart I/O Yönlendirmesi

- Java, standard input, output ve error I/O nesnelerini yönlendirmeyi de sağlar

```
FileInputStream in = new FileInputStream("/in.txt");
```

```
System.setIn(in);
```

```
FileOutputStream out = new FileOutputStream("/out.txt");
```

```
System.setOut(new PrintStream(out));
```

```
FileOutputStream err = new FileOutputStream("/err.txt");
```

```
System.setErr(new PrintStream(err));
```

# JVM Console

- **System.console()** metodu ile varsa JVM'e ilişkilendirilmiş mevcut Console nesnesi dönülür
- Eğer JVM **komut satırından** çalıştırılmış ise bir **Console** dönülecektir
- Eğer JVM bir **arka plan process'i** tarafından çalıştırılmış ise bu durumda bir **Console mevcut olmayacaktır**
- Böyle bir durumda ise **System.console()** **NULL** döner
- Console nesnesi vasıtası ile **veri okumak ve yazmak** oldukça kolaydır

# JVM Console

```
Console console = System.console();
String username = console.readLine("[%s]:", "Username");
char[] password = console.readPassword("[%s]", "Password");

boolean authenticated = authenticate(username, password);

if (!authenticated) {
    console.format("Wrong username or password [%s]", username);
} else {
    // ...
}
```

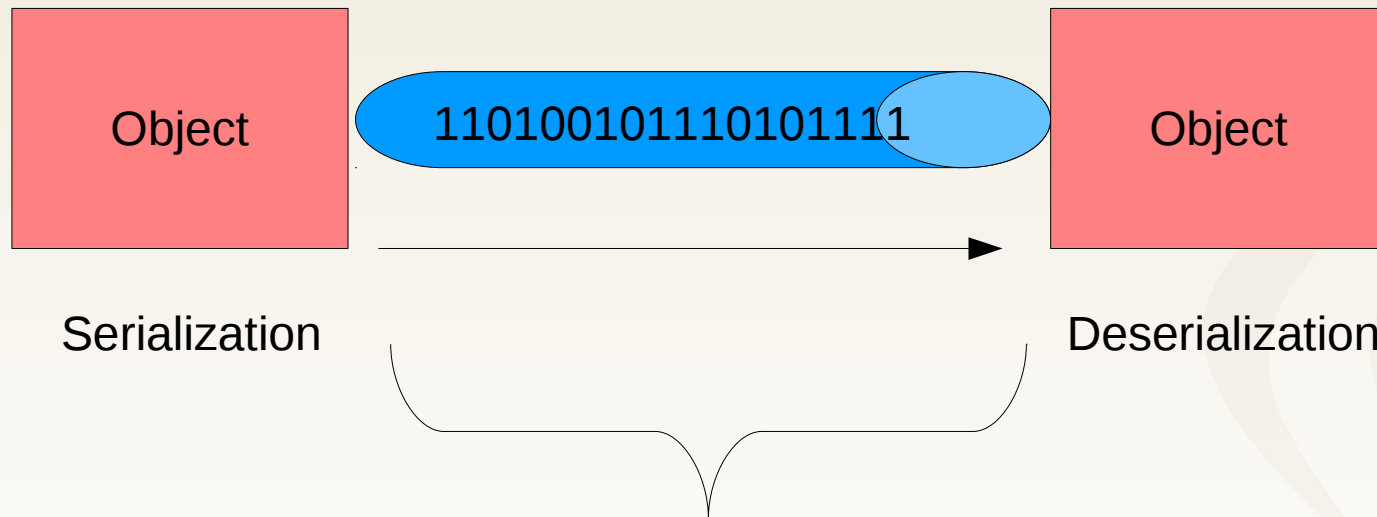
- Ayrıca Console **reader()** ve **writer()** metotları ile console ile ilişkilendirilmiş Reader ve Writer nesnelerine de erişilebilir



# Object Serialization İşlemi

- **Java nesnelerinin** network üzerinde taşınabilmeleri, dosya sistemine veya veritabanına yazılabilmeleri için **byte stream'e dönüştürülmeleri** gerekir
- Bu işleme **object serialization** adı verilir
- Benzer biçimde byte stream'in tekrar **Java nesnelere dönüştürülmesi** gerekir
- Bu işleme de **object deserialization** denir

# Object Serialization İşlemi



Binary data network üzerinde transfer edilebilir,  
dosya sistemine kaydedilebilir, veritabanınınında saklanabilir  
serialization/deserialization işlemi marshalling/unmarshalling  
olarak da bilinir

# Serializable Arayüzü

```
public class Personel implements Serializable {
    private int id;
    private String adi;
    private transient Date dogumTarihi;

    public Personel(int id, String adi, Date dogumTarihi) {
        this.id = id;
        this.adi = adi;
        this.dogumTarihi = dogumTarihi;
    }

    public int getId() {
        return id;
    }

    public String getAdi() {
        return adi;
    }

    public Date getDogumTarihi() {
        return dogumTarihi;
    }
}
```

Nesnenin serialization işlemine tabi tutulabilmesi için Serializable arayüzünü implement etmesi gerekir

Serialization işlemi sırasında bütün Property'ler serialization'a tabi tutulur Herhangi bir property'yi bu sürecin dışında tutmak için transient kullanılır

# ObjectOutputStream & ObjectInputStream Kullanımı

```
Personel p = new Personel(123, "Kenan Sevindik", new  
SimpleDateFormat("dd/MM/yyyy").parse("07/01/1976"));
```

```
ByteArrayOutputStream bos = new ByteArrayOutputStream();
```

```
ObjectOutputStream os = new ObjectOutputStream(bos);  
os.writeObject(p);
```

```
byte[] serData = bos.toByteArray();
```

Nesnenin serialized halini bir  
OutputStream'e yazmak ister

```
ByteArrayInputStream bin = new ByteArrayInputStream(serData);  
ObjectInputStream is = new ObjectInputStream(bin);
```

```
Personel p2 = (Personel)is.readObject();
```

```
System.out.println(p2);
```

Serialized data'yı bir  
InputStream'den okumak ister

# NotSerializableException Hatası

```
Exception in thread "main" java.io.NotSerializableException:  
com.javaegitimleri.Personel at  
java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:1164)  
at java.io.ObjectOutputStream.writeObject(ObjectOutputStream.java:330)  
at com.javaegitimleri.Formula1.main(Formula1.java:24)
```

- Serialization işlemi ile **bütün nesne hiyerarşisi byte formatına** dönüştürülür
- Hiyerarşideki bütün nesnelerin ya **Serializable** olması gerekir, ya da değişkenler **transient** ile tanımlanmış olmalıdır
- Eğer nesnenin sınıfı veya nesnenin içerisindeki değerlerden herhangi birisi **Serializable** değilse **NotSerializableException** fırlatılır

# Serial Version UID Değeri Ne İşe Yarar?

- JVM, Serializable sınıflar için sınıf içindeki field'lara bakarak otomatik olarak bir **serialization id** üretir
- Serialize edilmiş nesne verisi içerisinde bu **serialization id değeri** de saklanmaktadır
- Deserialization sırasında da bu değer ile sınıfın o anki **serialization id değeri karşılaştırılarak** eşleşip eşleşmediğine bakılır
- Eğer Serializable bir **sınıfın field'larında bir değişiklik meydana gelirse** (field ekleme/çıkarma vs) serialization id değeri değişir

# Serial Version UID Değeri Ne İşe Yarar?

- Böyle bir durumda da sınıfın önceki halinden yaratılıp serialize edilen nesneler **sınıfın yeni hali ile deserialize edilemez**
- Bunun önüne geçmek için serial version UID değeri manuel yönetilebilir
- Bu durumda **id değeri programcı tarafından değiştirilene kadar** serialization – deserialization çalışmaya devam edecektir

# Serial Version UID Örneği

```
public class Personel implements Serializable {
```

```
    private static final long serialVersionUID = 401045231236475483L;
```

```
    private int id;  
    private String adi;  
    private transient Date dogumTarihi;
```

```
    public Personel(int id, String adi, Date dogumTarihi) {  
        this.id = id;  
        this.adi = adi;  
        this.dogumTarihi = dogumTarihi;  
    }
```

```
    public int getId() {  
        return id;  
    }
```

```
    public String getAdi() {  
        return adi;  
    }
```

```
    public Date getDogumTarihi() {  
        return dogumTarihi;  
    }
```

```
}
```

Yukarıdaki değer IDE tarafından otomatik üretilmiştir. Programcı tarafından herhangi bir değer atanabilir



# İletişim



[www.harezmi.com.tr](http://www.harezmi.com.tr)

[www.java-egitimleri.com](http://www.java-egitimleri.com)



[info@harezmi.com.tr](mailto:info@harezmi.com.tr)

[info@java-egitimleri.com](mailto:info@java-egitimleri.com)



[@HarezmiBilisim](https://twitter.com/HarezmiBilisim)

[@JavaEgitimleri](https://twitter.com/JavaEgitimleri)