YMT 112 Algoritma ve Programlama

www.kriptarium.com/programlama

Geçen Hafta

- Java OOP Kavramları
  - Kapsülleme
  - Çok biçimlilik
    - Overloading
    - Overridding
  - Kalıtım
    - Extends, super
    - Abstract
    - Interface
- Java ve Dosya İşlemleri
- Java Hata Yönetimi

Olağan Dışı Durumların Değerlendirilmesi

- Bir programın çalışması esnasında normalde karşılaşılmaması gereken bir durum oluşursa, sektörde buna olağan dışı durum (exception) ve bu durumda ne yapılacağının belirlenmesine de olağan dışı durum yönetimi (exception handling) denir.
- Bir bölme işleminde paydanın 0 olması, beklenmeyen ya da olağan dışı bir durumdur.
- Aynı şekilde bir dosyaya erişim yapmak isteyen bir komutun o dosyayı bulamaması da olağan dışı bir durumdur.
- C++ diline kadar, olağan dışı durumların tespiti ve yö
- netimi yazılımcıların özel olarak geliştirdikleri kod parçalarıyla yapılabiliyordu. C++ dilinden itibaren olağan dışı durum yönetimi, dilin içine eklenen özel komut, sınıf ve metotlarla yapılabiliyor.
- Java ile birlikte, Microsoft Visual'te olağan dışı durum yönetimi, try / catch / finally yapısı ve ilgili sınıf ve metotlarla gerçekleştirilebiliyor.

try/catch/finally Yapısı

```
try
{
    //hesaplanmak istenen ifade
}
catch
{
    //Bir hata türü tespit edilince verilmesi gereken mesaj
}
catch
{
    //başka Bir hata türü tespit edilince verilmesi gereken mesaj
}
finally
{
    //her durumda çalıştırılacak olan kod parçası
}
```

```java
public class ExceptionOrnek1 {
    public static void main(String[] args) {

        String str = "Bunu dosyaya yazdir";
        File file = new File("dosya.txt");
        FileWriter fileWriter = null;

        try {
            fileWriter = new FileWriter(file, false);
            fileWriter.write(str);
        }
        catch (IOException dosyaHatasi) {
            dosyaHatasi.printStackTrace();
        }
        finally {
            try {
                if (fileWriter != null)
                    fileWriter.close();
            }
            catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

Kendi hatalarımızı oluşturma

- Program içerisinde kendi hazırladığımız sınıf ve metodlar zaman zaman bize özgü hatalar fırlatabilirler. Bunu sağlamak için kendimize ait bir hata sınıfı oluşturabiliriz. Java Exception sınıfı extend edilebilir bir sınıf olduğundan aşağıdaki gibi bir hata sınıfı hazırlanabilir:

```
public class OzelException extends Exception {
    private static final long serialVersionUID = 1L;

    @Override
    public String getMessage() {
        return "Benim hatam olustu.";
    }
}
```

```java
public class ExceptionOrnek2 {
    public static void main(String[] args) {
        try {
            hataFirlatir();
        }
        catch (OzelException e) {
            System.out.println(e.getMessage());
        }
    }

    public static void hataFirlatir() throws OzelException {
        throw new OzelException();
    }
}
```

- Hata Tipleri: Hatanın Anlamı
- ArithmeticException Aritmatiksel bir işlem sırasında oluşan hata
- ArrayIndexOutOfBoundsException Array sınırlarından çıkılma hatası
- ArrayStoreException Array yapısına uymayan bir değer kaydedilmeye çalışıldığında
- ClassCastException Uygunsuz class atama işlemi
- IllegalArgumentException Uygun olmayan argüman kullanımı
- IllegalMonitorStateException Uygun olmayan monitör işlemi
- IllegalStateException Sistemin uygun pozisyonda olmadığını belirtir
- IllegalThreadStateException Tread durumunun uygun olmadığını belirtir
- IndexOutOfBoundsException Index'in sınırlardan çıkılma hatası
- NegativeArraySizeException Array'in boyutuna negatif değer verilmiş olması hatası
- NullPointerException Boş pointer hatası
- NumberFormatException Uygun olmayan sayı formatı hatası
- SecurityException Güvenlik hatası
- StringIndexOutOfBounds String yapısının sınırlarından çıkma hatası
- UnsupportedOperationException Desteklenmeyen işlem hatası
- ClassNotFoundException Class bulunmama hatası
- CloneNotSupportedException Çoğalmaya çalışılan bir objenin çoğalamayacağını belirten hata
- IllegalAccessException Uygunsuz erişim hatası
- InstantiationException Bir objenin oluşturulması esnasında oluşan hata
- InterruptedException Bir tread'in diğer tread'i durdurma hatası
- NoSuchFieldExceptionAranılan alanın olmadığı hatası
- NoSuchMethodException Kıllanılan methodun bulunmama hatası

```
public class Program {
    public static void main(String[] args) {

        try {
            // Divide by zero.
            int value = 1 / 0;
        } catch (Exception ex) {
            // Display exception.
            System.out.println(ex);
        }
    }
}
```

Output

```
java.lang.ArithmeticException: / by zero
```

```
public class Program {
    public static void main(String[] args) {

        // An input string.
        String name = "sam";
        System.out.println(name.length());

        // When string is null, an exception occurs.
        name = null;
        System.out.println(name.length());
    }
}
```

Output

```
3
Exception in thread "main" java.lang.NullPointerException
    at program.Program.main(Program.java:9)
Java Result: 1
```

Java program that causes compilation error

```
public class Program {
    public static void main(String[] args) {
        throw new Exception();
    }
}
```

Output

```
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
    Unhandled exception type Exception
    at program.Program.main(Program.java:5)
```

Java program that has no compilation error

```
public class Program {
    public static void main(String[] args) throws Exception {
        throw new Exception();
    }
}
```

Output

```
Exception in thread "main" java.lang.Exception
    at program.Program.main(Program.java:5)
```

```java
public class Program {
    public static void main(String[] args) {

        String value = null;
        try {
            // This statement causes an exception because value is null.
            // ... Length() requires a non-null object.
            System.out.println(value.length());
        } catch (Exception ex) {
            // This runs when the exception is thrown.
            System.out.println("Exception thrown!");
        } finally {
            // This statement is executed after the catch block.
            System.out.println("Finally done!");
        }
    }
}
```

Output

```
Exception thrown!
Finally done!
```

```
public class Program {
    public static void main(String[] args) {

        try {
            System.out.println("In try");
        } finally {
            // The finally is run even if no exception occurs.
            System.out.println("In finally");
        }
        System.out.println("...Done");
    }
}
```

Output

```
In try
In finally
...Done
```

```
public class Program {
    public static void main(String[] args) {

        long t1 = System.currentTimeMillis();

        // Version 1: check against zero before division.
        for (int i = 0; i < 1000000; i++) {

            int v = 0;
            for (int x = 0; x < 10; x++) {
                if (x != 0) {
                    v += 100 / x;
                }
            }
            if (v == 0) {
                System.out.println(v);
            }
        }

        long t2 = System.currentTimeMillis();
```

```
        long t2 = System.currentTimeMillis();

        // Version 2: handle exceptions when divisor is zero.
        for (int i = 0; i < 1000000; i++) {

            int v = 0;
            for (int x = 0; x < 10; x++) {
                try {
                    v += 100 / x;
                } catch (Exception ex) {
                    // Errors are encountered.
                }
            }
            if (v == 0) {
                System.out.println(v);
            }
        }

        long t3 = System.currentTimeMillis();

        // ... Times.
        System.out.println(t2 - t1);
        System.out.println(t3 - t2);
    }
}
```

Results

```
36 ms:   if-check
89 ms:   try, catch
```

```
public class Program {

    static void applyRecursion() {
        applyRecursion();
    }

    public static void main(String[] args) {
        // Begin recursion.
        applyRecursion();
    }
}
```

Output

```
Exception in thread "main" java.lang.StackOverflowError
    at program.Program.applyRecursion(Program.java:6)
    at program.Program.applyRecursion(Program.java:6)
    at program.Program.applyRecursion(Program.java:6)...
```

Dosya G/Ç İşleminin Gerekliliği

- Program sona erdiğinde kullanılan veriler kaybolur.
- Verileri kaybetmemek için dosyada saklanması gereklidir.
- Aynı şekilde klavyeden girilen verilerin de program çalıştırıldığında tekrar tekrar girilmesi yerine kaydedilip tekrar çalıştırıldığında okunarak elde edilmesi gerekir.

Dosya İşlemleri

- import java.io.File;
- File dosya = new File(dosyaAdi);
  - dosya.getAbsolutePath()
  - dosya.getPath()
  - dosya.getName()
  - dosya.getParent()
  - dosya.exists()
  - dosya.canRead()
  - dosya.canWrite()
  - dosya.isDirectory()
  - dosya.isFile()
  - dosya.lastModified()
  - dosya.length()

Yeni Dosya Oluşturma

```
File f = new File(dosyaAdi); // Dosya nesnesi
if(!f.exists()){ //Dosya zaten var mı
        f.createNewFile(); //Dosyayı oluştur
}
```

```
File dosya = new File("ornek.dat");
```

Bulunulan klasordeki ornek.dat dosyasını açar

```
File dosya = new File
        ("C:/OrnekProgram/test.dat");
```

C:\OrnekProgram klasöründeki test.dat dosyasını açar.
Dosyanın adresi / ayracı ile verilir.

```
if ( dosya.exists( ) ) {
```

dosya değişkeni gerçekten var olan bir dosyayı mı gösteriyor.

```
if ( dosya.isFile() ) {
```

dosya bir doysa mı yoksa bir klasör mü.

```
File klasor = new
        File("C:/Programlarım/java");

String dosyalar[] = klasor.list();

for (int i = 0; i < dosyalar.length; i++) {
    System.out.println(dosyalar[i]);
}
```

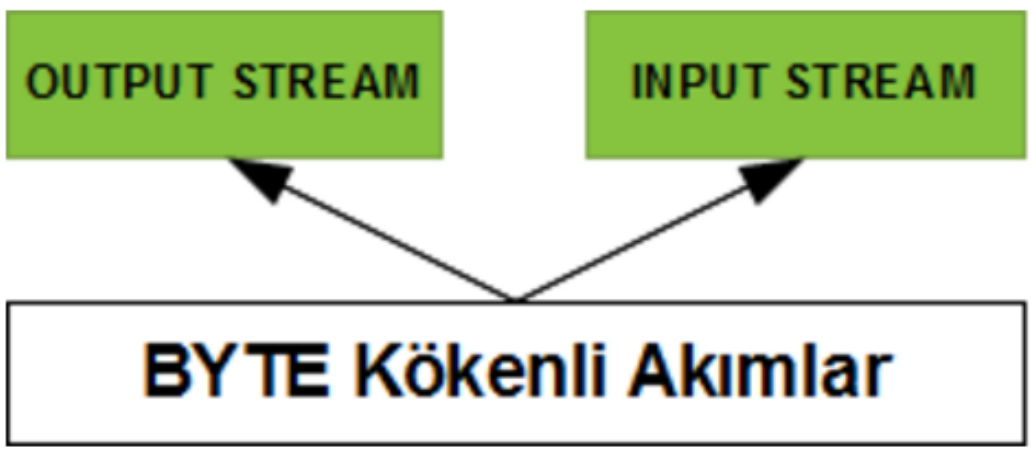
C:\Programlarım\java verilen klasördeki bütün dosyaları listeler.

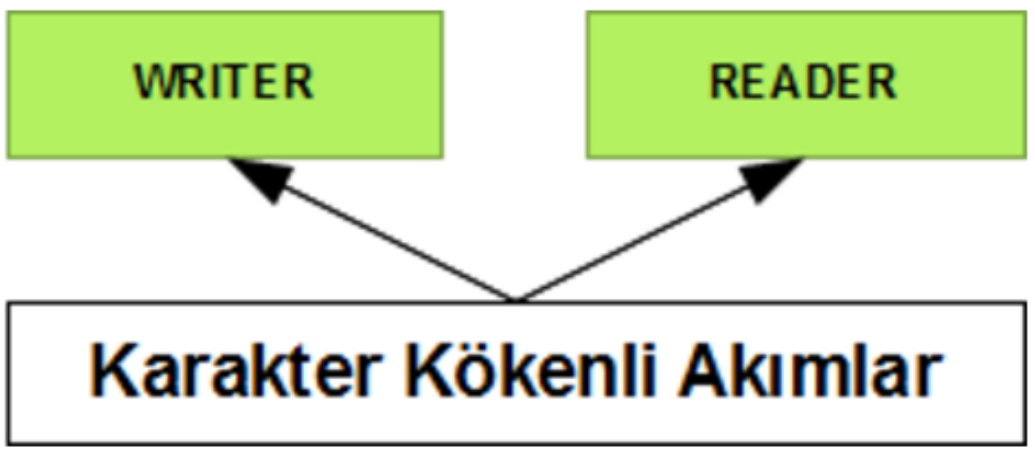
GİRİŞ-ÇIKIŞ AKIMLARI

Java'da Giriş-Çıkış akımları, iki ana gruba ayrılır:

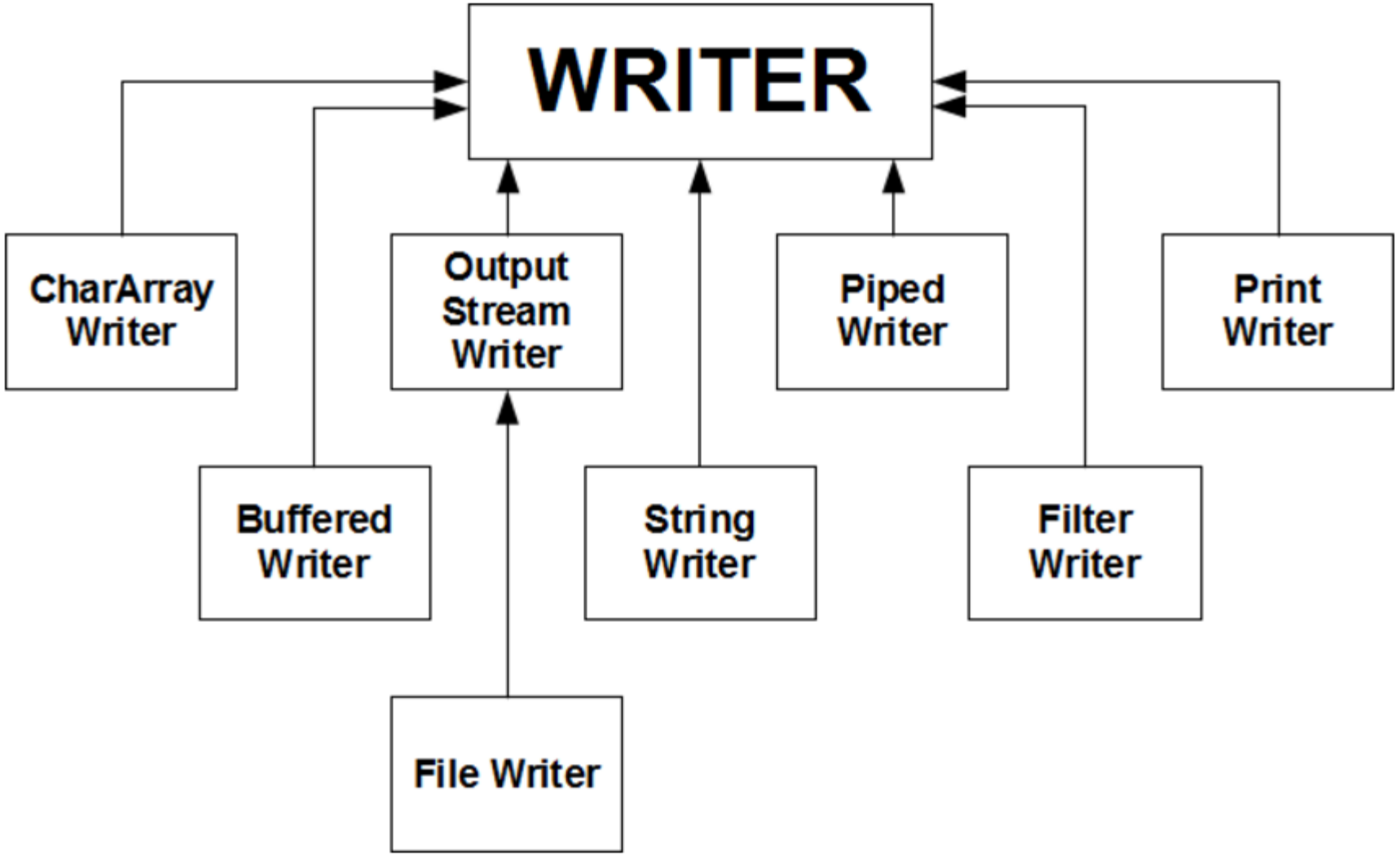
1. Karakter kökenli akımlar
2. Byte kökenli akımlar

Bunların herbiri için ayrı sınıflar tanımlanmıştır. Aşağıdaki şekil bu konuda bize fikir veriyor:





Karakter Kökenli Akımlar



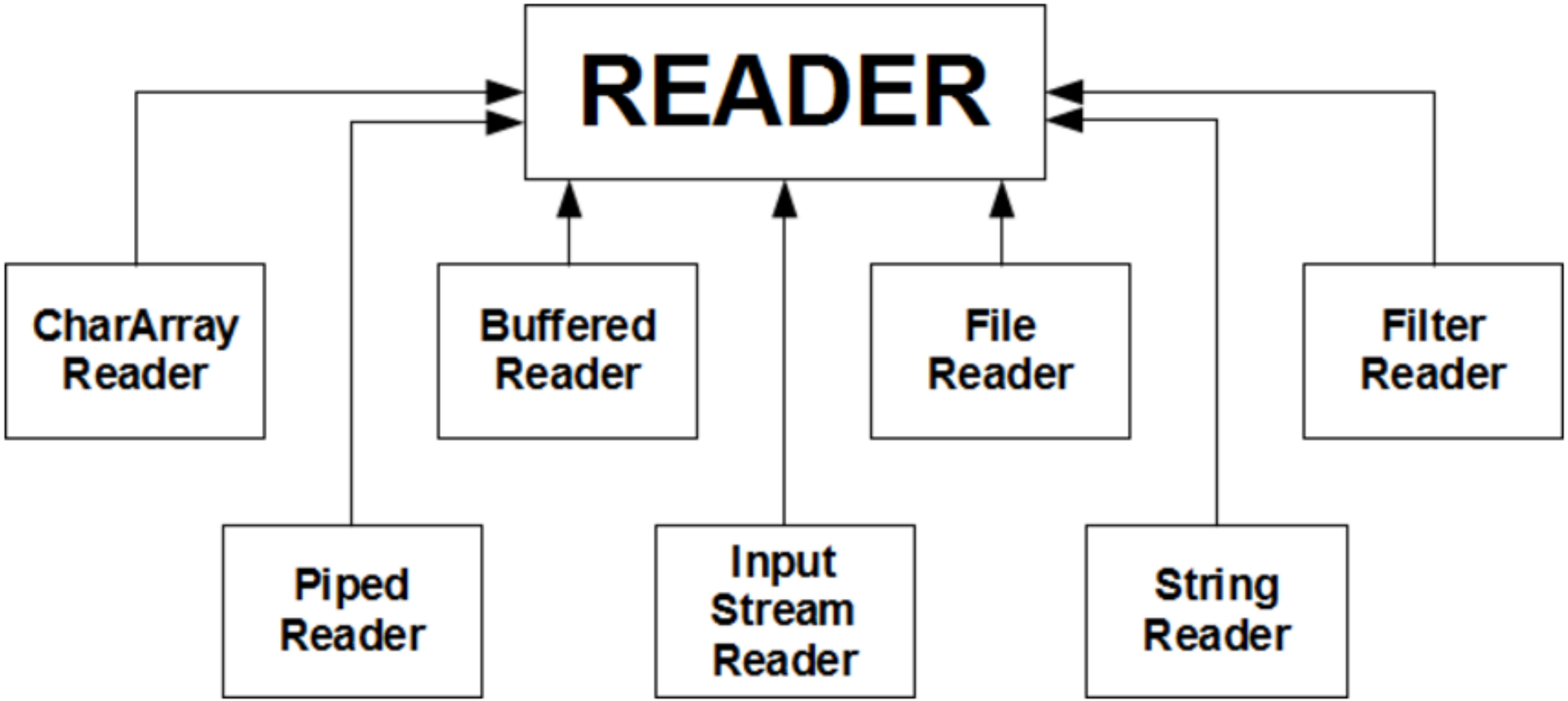
FileWriter Sınıfının Kullanımı

Aşağıdaki örnek Java programı, *FileWriter* sınıfından **yazar** adlı bir nesne oluşturarak, bu nesne yardımıyla **siir.txt** adlı *text* tipi dosyaya bilgi kaydediyor. Dosyaya bir satırlık bilgi *write* metoduyla yazılıyor ve satır sonu '**\n**' karakteriyle belirleniyor.

```
1  import java.io.FileWriter;
2  import java.io.IOException;
3
4  public class TextDosyaYaz {
5      public static void main(String[] args) throws IOException {
6          String dosyaAdi = "siir.txt";
7          FileWriter yazar = new FileWriter(dosyaAdi);
8          yazar.writeln("coklarindan dusuyor da bunca");
9          yazar.writeln("gormuyor gelip gecenler");
10         yazar.writeln("egilip aliyorum");
11         yazar.writeln("solgun bir gul oluyor dokununca - behcet necatigil");
12         yazar.close();
13     }
14 }
```

Metin Türü Dosyadan Bilgi Okumak

Metin türü bir dosyadan bilgi okumak için *Reader* sınıfını ve alt sınıflarını kullanırız. Aşağıda Reader sınıfı ve ilgili sınıfların hiyerarşisini görüyoruz:



Dosyaya Veri Yazma

```
1
2
3           String str = "Bunu dosyaya yazdir";
4
5           File file = new File("dosya.txt");
6           if (!file.exists()) {
7               file.createNewFile();
8           }
9
10          FileWriter fileWriter = new FileWriter(file, false);
11          BufferedWriter bWriter = new BufferedWriter(fileWriter);
12          bWriter.write(str);
13          bWriter.close();
14
15
```

Dosyadan Veri Okuma

```
1
2
3   FileReader fileReader = new FileReader(file);
4   String line;
5
6   BufferedReader br = new BufferedReader(fileReader);
7
8   while ((line = br.readLine()) != null) {
9
10      System.out.println(line);
11
12  }
13
14  br.close();
15
16
```

Bir önceki örnekte **siir.txt** adı ile oluşturduğumuz dosyayı şimdi bir Java programı ile okuyarak ekrana aktaracağız:

```
1  import java.io.BufferedReader;
2  import java.io.FileReader;
3  import java.io.IOException;
4
5  public class TextDosyaOku {
6      public static void main (String[] args){
7          String dosyaAdi = "siir.txt";
8          String satir;
9          try{
10             BufferedReader oku = new BufferedReader(new FileReader(dosyaAdi));
11             satir = oku.readLine();
12             while (satir != null) {
13                 System.out.println(satir);
14                 satir = oku.readLine();
15             }
16             oku.close();
17         }
18         catch (IOException iox){
19             System.out.println(dosyaAdi+" adli dosya okunamiyor.");
20         }
21     }
22 }
```

```java
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Kopyala{
    String kaynakDosyaAdi, hedefDosyaAdi;
    BufferedReader kaynakBuffer;
    BufferedWriter hedefBuffer;
    String satir;

    public static void main (String[] args) {
        if (args.length == 3 && args[1].toUpperCase().equals("TO")) {
            new Kopyala().kopya(args[0], args[2]);
        }
        else {
            System.out.println("Kullanim sekli: ");
            System.out.println("java Kopyala <dosya1> to <dosya2>");
        }
    }
```

```java
public boolean kopya(String isim1, String isim2) {
    kaynakDosyaAdi=isim1;
    hedefDosyaAdi=isim2;
    return dosyayiAc() && dosyayiKopyala() && dosyayiKapat();
}
```

```
private boolean dosyayiAc() {
    // Kaynak dosyayı aç
    try {
        kaynakBuffer = new BufferedReader(
                new FileReader(kaynakDosyaAdi));
    }
    catch (IOException e) {
        System.out.println(kaynakDosyaAdi
                +" dosyasinin acilisinda sorun var!");
        return false;
    }

    // Hedef dosyayı aç
    try    {
        hedefBuffer = new BufferedWriter(
                new FileWriter(hedefDosyaAdi));
    }
    catch (IOException e) {
        System.out.println(hedefDosyaAdi
                +" dosyasinin acilisinda sorun var!");
        return false;
    }
    return true; //iki dosya da ulasilabilirse true döndürür
```

```
private boolean dosyayiKopyala() {
    //esas metodumuz
    try    {
        satir = kaynakBuffer.readLine();
        while (satir != null) {
            hedefBuffer.write(satir);
            hedefBuffer.newLine();
            satir = kaynakBuffer.readLine();
        }
    }
    catch (IOException e) {
        System.out.println("Kopyalama yapilamiyor.");
        return false;
    }
    return true;
}
```

```
private boolean dosyayiKapat() {
    boolean sorunsuzMu=true;
    // kaynağı kapat
    try    {
        kaynakBuffer.close();
    }
    catch (IOException e) {
        System.out.println(kaynakDosyaAdi+" dosyasinin kapanisinda hata!");
        sorunsuzMu = false;
    }
    // hedefi kapat
    try    {
        hedefBuffer.close();
    }
    catch (IOException e) {
        System.out.println(hedefDosyaAdi+" dosyasinin kapanisinda hata!");
        sorunsuzMu = false;
    }
    return sorunsuzMu;
}
```

```
import java.io.BufferedInputStream;
import java.io.InputStream;
import java.net.URL;
import java.net.URI;

public class Program {
    public static void main(String[] args) throws Exception {

        // Create URI and URL objects.
        URI uri = new URI("http://en.wikipedia.org/wiki/Main_Page");
        URL url = uri.toURL();
        InputStream in = url.openStream();

        // Used a BufferedInputStream.
        BufferedInputStream reader = new BufferedInputStream(in);

        // Read in the first 200 bytes from the website.
        byte[] data = new byte[200];
        reader.read(data, 0, 200);

        // Convert the bytes to a String.
        String result = new String(data);
        System.out.println(result);
    }
}
```

```
import java.io.IOException;
import java.nio.file.*;

public class Program {
    public static void main(String[] args) throws IOException {

        // Get paths for input and target files.
        FileSystem system = FileSystems.getDefault();
        Path original = system.getPath("C:\\programs\\file.txt");
        Path target = system.getPath("C:\\programs\\file-2.txt");

        // Copy original to target location.
        Files.copy(original, target);

        // Helpful message.
        System.out.println("File copied!");
    }
}
```

Output

```
File copied!
```

```
import java.io.IOException;
import java.nio.file.*;

public class Program {
    public static void main(String[] args) throws IOException {

        // Get paths.
        Path original = FileSystems.getDefault().getPath(
                "C:\\programs\\file.txt");
        Path target = FileSystems.getDefault().getPath(
                "C:\\programs\\file-2.txt");

        // Replace an existing file with REPLACE_EXISTING.
        // ... No FileAlreadyExistsException will be thrown.
        Files.copy(original, target, StandardCopyOption.REPLACE_EXISTING);

        System.out.println("DONE");
    }
}
```

Output

```
DONE
```

```
import java.io.IOException;
import java.nio.file.*;

public class Program {
    public static void main(String[] args) {

        // These files do not exist in our example.
        FileSystem system = FileSystems.getDefault();
        Path original = system.getPath("C:\\programs\\mystery.txt");
        Path target = system.getPath("C:\\programs\\mystery-2.txt");

        try {
            // Throws an exception if the original file is not found.
            Files.copy(original, target, StandardCopyOption.REPLACE_EXISTING);
        } catch (IOException ex) {
            System.out.println("ERROR");
        }
    }
}
```

Output

```
ERROR
```

OOP Örnekleri

```
class Animal {
    public void scratch() {
        System.out.println("Animal scratched");
    }
}

class Cat extends Animal {
    public void scratch() {
        // Call super-class method.
        super.scratch();
        System.out.println("Cat scratched");
    }
}

public class Program {
    public static void main(String[] args) {

        // Create cat and call method.
        Cat cat = new Cat();
        cat.scratch();
    }
}
```

```
class Page {
    public void test() {
        // This class inherits from Object only.
        // ... So it gains access to methods like hashCode on Object.
        System.out.println(super.hashCode());
    }
}

public class Program {
    public static void main(String[] args) {
        Page page = new Page();
        page.test();
    }
}
```

Output

```
366712642
```

```
class Data {
    public Data() {
        System.out.println("Data constructor");
    }
}

class Image extends Data {
    public Image() {
        // Call the superclass constructor.
        super();
    }
}

public class Program {
    public static void main(String[] args) {
        Image value = new Image();
        System.out.println("Done");
    }
}
```

```java
abstract class Page {
    public abstract void open();
}

class Article extends Page {
    public void open() {
        System.out.println("Article.open");
    }
}

class Post extends Page {
    public void open() {
        System.out.println("Post.open");
    }
}

public class Program {
    public static void main(String[] args) {

        // We cannot directly create a Page.
        Page page = new Article();
        page.open();

        Page page2 = new Post();
        page2.open();
    }
}
```

```java
public class Box {

    int width;
    int height;

    public Box(int width, int height) {
        // Store arguments as fields.
        this.width = width;
        this.height = height;
    }

    public int area() {
        // Return area.
        return this.width * this.height;
    }
}
```

```java
public class Program {
    public static void main(String[] args) {
        // Create box.
        Box box1 = new Box(2, 3);
        int area1 = box1.area();

        // Create another box.
        Box box2 = new Box(1, 5);
        int area2 = box2.area();

        // Display areas.
        System.out.println(area1);
        System.out.println(area2);
    }
}
```

```
class A {
    public void call() {
        System.out.println("A.call");
    }
}

class B extends A {
    public void call() {
        System.out.println("B.call");
    }
}

public class Program {
    public static void main(String[] args) {

        // Reference new B class by A type.
        A a = new B();
        // Invoke B.call from A class instance.
        a.call();
    }
}
```

```java
class Animal {
    public void breathe() {
        System.out.println("Breathe");
    }
}

class Dog extends Animal {
    public void bark() {
        System.out.println("Woof");
    }
}

public class Program {
    public static void main(String[] args) {
        // On an animal we can call one method.
        Animal animal = new Animal();
        animal.breathe();

        // On a dog we can use Animal or Dog methods.
        Dog dog = new Dog();
        dog.breathe();
        dog.bark();
    }
}
```

```
class Box {
    // This class has final ints on it.
    final int width = 10;
    final int height = width * 2;

    public int area() {
        return width * height;
    }
}

public class Program {

    public static void main(String[] args) {
        // Create and use class with final ints.
        Box box = new Box();
        System.out.println(box.area());
    }
}
```

```
public class Program {

    // This String is both final and static.
    final static String description = "Hello world program";

    public static void main(String[] args) {
        // Print string.
        System.out.println(description);
    }
}
```

```
public class Program {
    public static void main(String[] args) {

        // This is a final String.
        final String name = "Hyperion";

        // Construct another String.
        String test = "Hyper";
        String test2 = test + "ion";

        System.out.println(name);

        // Equals test the two strings.
        System.out.println(name.equals(test2));

        // The string references are different.
        System.out.println(name == test2);
    }
}
```

```
public class Program {
    public static void main(String[] args) {

        final int size = 10;
        // This fails with a compilation error.
        size = 200;
    }
}
```

```
interface Site {
    public void display();
}

class Ruby implements Site {
    public void display() {
        System.out.println("Ruby.display");
    }
}

class Java implements Site {
    public void display() {
        System.out.println("Java.display");
    }
}

public class Program {
    public static void main(String[] args) {

        // Use implementations through an interface.
        Site site = new Ruby();
        site.display();

        Site site2 = new Java();
        site2.display();
    }
}
```

```
import java.util.ArrayList;
import java.util.Collection;
import java.util.Vector;

public class Program {
    public static void main(String[] args) {

        ArrayList<Integer> list = new ArrayList<>();
        list.add(1);

        Vector<Integer> vec = new Vector<>();
        vec.add(2);
        vec.add(3);

        // Treat the collections by their interface.
        display(list);
        display(vec);
    }

    static void display(Collection<Integer> col) {
        // Display size of collection.
        System.out.println(col.size());
    }
}
```

```java
interface Box {
    public int area(boolean accurate);
}

interface Cube {
    public int volume(boolean accurate);
}

class Unit implements Box, Cube {
    public int area(boolean accurate) {
        if (accurate) {
            return 4;
        } else {
            return 1;
        }
    }

    public int volume(boolean accurate) {
        if (accurate) {
            return 100;
        } else {
            return 10;
        }
    }
}
```

```java
public class Program {
    public static void main(String[] args) {

        // Create class and call methods.
        Unit unit = new Unit();
        System.out.println(unit.area(true));
        System.out.println(unit.volume(false));
    }
}
```

```
import java.util.ArrayList;
import java.util.Collection;

public class Program {
    public static void main(String[] args) {

        // Use Collection interface with ArrayList.
        Collection<String> coll = new ArrayList<>();
        coll.add("cat");

        // Cast Collection to ArrayList.
        ArrayList<String> list = (ArrayList<String>) coll;
        System.out.println(list.size());
    }
}
```

```
interface Coffee {
    public void brew(int size) throws Exception;
}

class MediumRoast implements Coffee {
    public void brew(int size) throws Exception {
        // Throw an exception if size is too big.
        if (size > 100) {
            throw new Exception("Too much coffee");
        }
    }
}

public class Program {
    public static void main(String[] args) throws Exception {
        Coffee cup = new MediumRoast();
        cup.brew(1000);
    }
}
```

Output

```
Exception in thread "main" java.lang.Exception: Too much coffee
    at program.MediumRoast.brew(Program.java:12)
    at program.Program.main(Program.java:20)
```

```
import java.util.HashMap;

interface Box {
    public void test();
}

class CardboardBox implements Box {
    public void test() {
        System.out.println("cardboard");
    }
}

class PlasticBox implements Box {
    public void test() {
        System.out.println("plastic");
    }
}
```

```
public class Program {
    public static void main(String[] args) {

        HashMap<String, Box> map = new HashMap<>();

        // ... Put new class instances in HashMap.
        map.put("ship-100", new CardboardBox());
        map.put("pack-200", new PlasticBox());

        // ... Get interface reference from the HashMap.
        //     Call test on it.
        Box b = map.get("ship-100");
        if (b != null) {
            b.test();
        }
    }
}
```

```
interface Tester {
    public void test(int value) throws Exception;
}

class Perl implements Tester {
    public void test(int value) throws Exception {
        // Some useless logic.
        if (value < 0) {
            throw new Exception();
        }
    }
}
```

```
public class Program {
    public static void main(String[] args) throws Exception {

        Tester test = new Perl();
        Perl perl = (Perl) test;

        long t1 = System.currentTimeMillis();

        // ... Version 1: call interface method.
        for (int i = 0; i < 10000000; i++) {
            test.test(i);
        }

        long t2 = System.currentTimeMillis();

        // ... Version 2: call class method.
        for (int i = 0; i < 10000000; i++) {
            perl.test(i);
        }

        long t3 = System.currentTimeMillis();

        // ... Times.
        System.out.println(t2 - t1);
        System.out.println(t3 - t2);
    }
}
```

Output

```
27 ms, Interface method call
 8 ms, Non-interface method call
```