

# AĞ PROGRAMLAMA

## DERS 2:

- Programlama Temelleri
  - I/O İşlemleri
  - GUI Programlama
  - Multi-Thread Progrmalama

# Programlama Platformu ve Diğer Kaynaklar

---

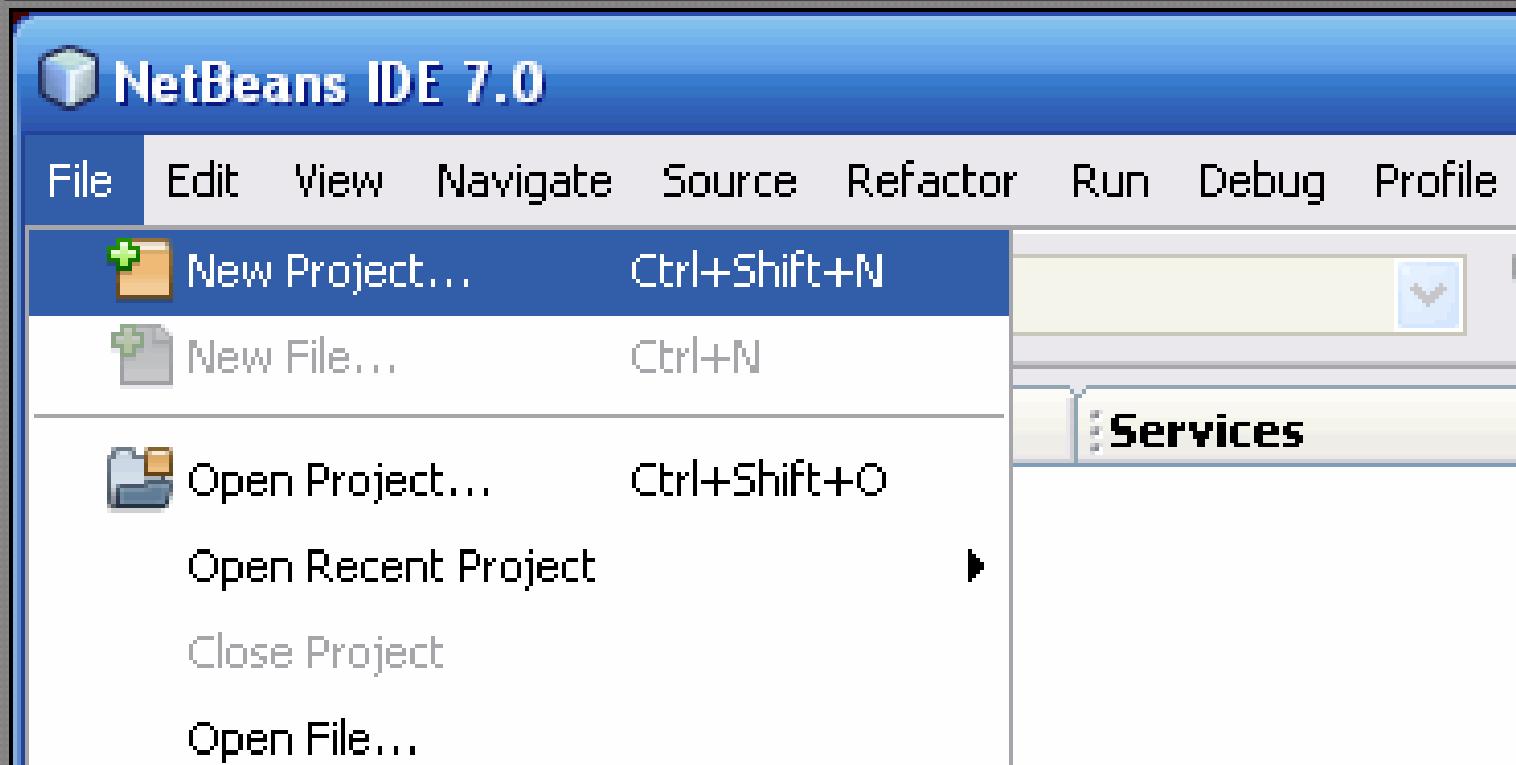
- JAVA projelerini geliştirmek için
  - JDK
    - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
  - NetBEANS (IDE)
    - <http://netbeans.org/downloads/index.html>
  - Eclipse (IDE)
    - <http://www.eclipse.org/downloads/>
- Eğitimler
  - <http://docs.oracle.com/javase/tutorial/>

# NetBeans IDEsi ile Programlamaya Giriş

---

- «Hello Word » uygulaması
  - Yeni Bir Proje oluşturma
  - Otomatik üretilmiş kod dosyasına ekleme yapma
  - Derleme ve .class dosyasının oluşturulması
  - Programın çalıştırılması

# Yeni Bir Proje oluşturma



# Yeni Bir Proje oluşturma

 New Project X

**Steps**

1. Choose Project  
2. ...

**Choose Project**

**Categories:**

- Java
- Maven
- NetBeans Modules
- Samples

**Projects:**

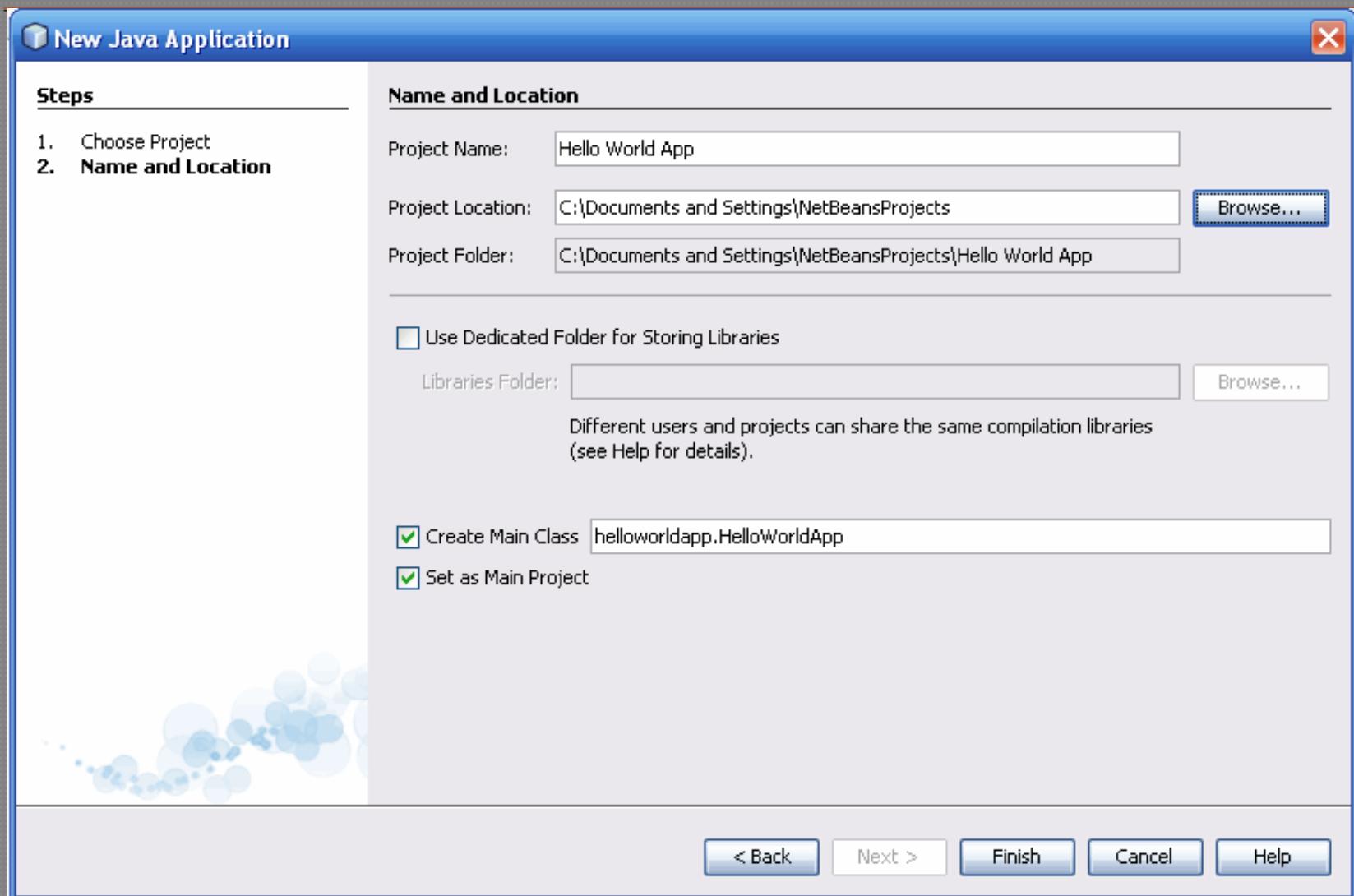
- Java Application
- Java Desktop Application
- Java Class Library
- Java Project with Existing Sources
- Java Free-Form Project

**Description:**

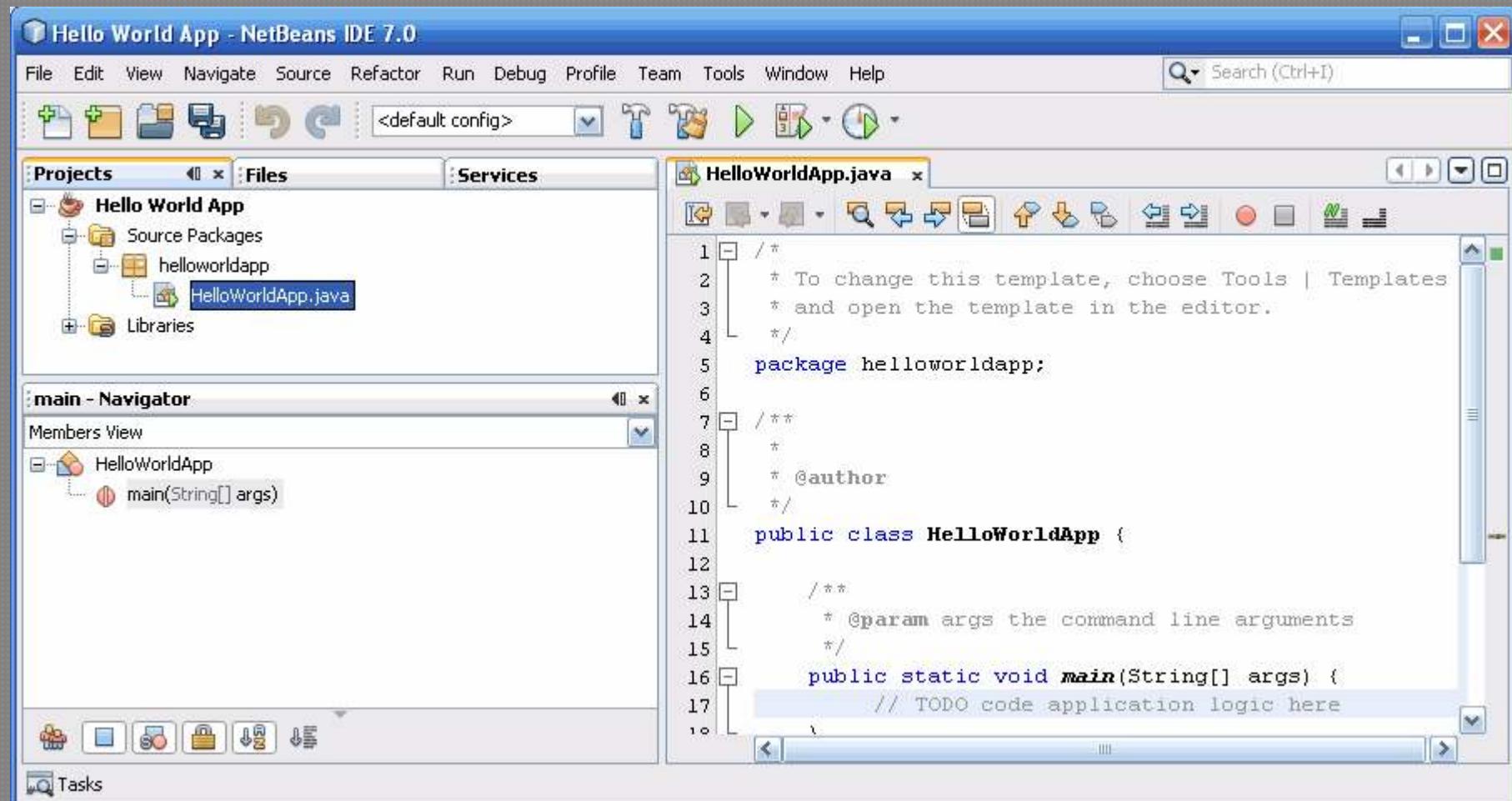
**Creates a new Java SE application** in a standard IDE project. You can also generate a main class in the project. Standard projects use an **IDE-generated Ant build script** to build, run, and debug your project.

[< Back](#) [Next >](#) [Finish](#) [Cancel](#) [Help](#)

# Yeni Bir Proje oluşturma



# Otomatik üretilmiş kod dosyasına ekleme yapma



# Otomatik üretilmiş kod dosyasına ekleme yapma

---

- Kod içerisindeki:

```
// TODO code application logic here
```

- Aşağıdaki ile değiştirelim

```
System.out.println("Hello World!"); // Display the string.
```

# Derleme ve .class dosyasının oluşturulması

- Projeyi derlemek için IDE'nin üst menüsünden Run | Build Main Project seçilir.
- Çıktı(output) ekranı aşağıdaki gibi olacaktır.

```
Output - Hello_World_App (clean,jar) Tasks
init:
deps-jar:
Created dir: C:\Documents and Settings\NetBeansProjects\Hello World App\build
Updating property file: C:\Documents and Settings\NetBeansProjects\Hello World App\build\built-jar.properties
Created dir: C:\Documents and Settings\NetBeansProjects\Hello World App\build\classes
Created dir: C:\Documents and Settings\NetBeansProjects\Hello World App\build\empty
Created dir: C:\Documents and Settings\NetBeansProjects\Hello World App\build\generated-sources\ap-source-output
Compiling 1 source file to C:\Documents and Settings\NetBeansProjects\Hello World App\build\classes
compile:
Created dir: C:\Documents and Settings\NetBeansProjects\Hello World App\dist
Copying 1 file to C:\Documents and Settings\NetBeansProjects\Hello World App\build
Nothing to copy.
Building jar: C:\Documents and Settings\NetBeansProjects\Hello World App\dist\Hello World App.jar
To run this application from the command line without Ant, try:
java -jar "C:\Documents and Settings\NetBeansProjects\Hello World App\dist\Hello World App.jar"
jar:
BUILD SUCCESSFUL (total time: 11 seconds)
```

# Programın Çalıştırılması

---

- IDE'nin menüsünden Run | Run Main Project seçilir.

The screenshot shows an IDE's output window titled "Output - Hello World App (run)". The window contains the following text:  
run:  
Hello World!  
BUILD SUCCESSFUL (total time: 6 seconds)

The window has a toolbar on the left with four icons: a green play button, a yellow play button, a brown square, and a wrench/cog icon. The title bar has standard window controls (minimize, maximize, close).

# Temel I/O İşlemleri

---

- I/O işlemleri için genel kütüphane
  - java.io
- Dosya I/O için
  - java.nio.file
- I/O Streamleri
  - Byte Streams
  - Character Streams
  - Buffered Streams
  - Scanning and Formatting
  - I/O from the Command Line
  - Data Streams
  - Object Streams

# Byte Streams

---

- 8bit'lik formlarda giriş ve çıkış işlemleri yapmak için kullanılır.
- Tüm byte streamleri InputStream ve OutputStream soyundan gelmektedir.
- Birçok byte stream java I/O işlemleri içerisinde mevcuttur. Örnek olarak dosya işlemlerini byte formatında gerçekleştirebileceğimiz FileInputStream ve FileOutputStream ele alınabilir.

# Byte Stream Örnek: xanadu.txt dosyasından byte byte kopyalama yapma

---

```
import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;

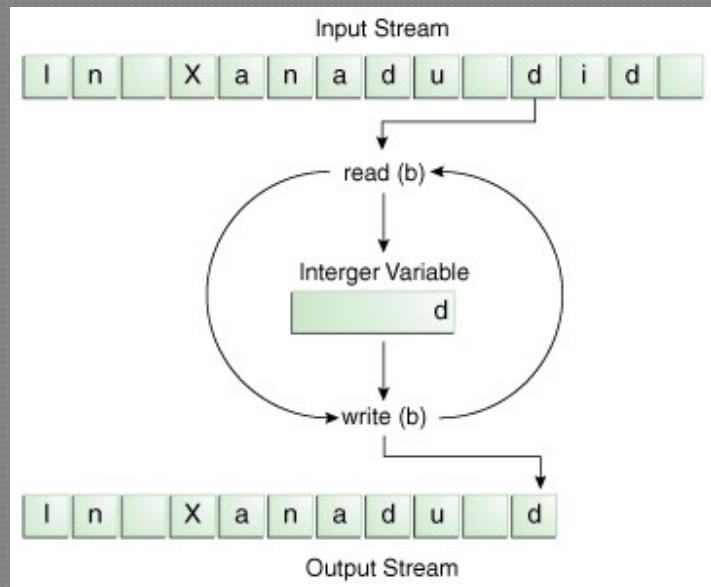
public class CopyBytes {
    public static void main(String[] args) throws IOException {

        FileInputStream in = null;
        FileOutputStream out = null;

        try {
            in = new FileInputStream("xanadu.txt");
            out = new FileOutputStream("outagain.txt");
            int c;

            while ((c = in.read()) != -1) {
                out.write(c);
            }
        } finally {
            if (in != null) {
                in.close();
            }
            if (out != null) {
                out.close();
            }
        }
    }
}
```

# Byte Stream Örnek: xanadu.txt dosyasından byte byte kopyalama yapma



Genel Sistem işleyışı.

Bir stream ile işiniz bittiğinde kapatmayı unutmayın.  
(Try catch finally bloğuna dikkat ediniz.)

# Character Streams

---

- Yapılacak I/O işlemlerinin 8bit ASCII kodlaması ile yapılmasını sağlarlar.
- Bir çok uygulama için byte streamlerle benzer özelliktidir. Daha karmaşık bir yapısı yoktur.
- Byte stream yerine Character streami kullanan uygulamaların yerel dile göre düzenlenmesi daha kolaydır. Genelleştirmesi için kolaylık sağlar.
- Reader ve Writer soyundan türetilmişlerdir.
- Örnek olarak FileReader ve FileWriter ile yapılmış örneği inceleyelim.

# Character Streams

---

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

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

        FileReader inputStream = null;
        FileWriter outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

# Character Streams

---

- CopyCharacter programı CopyByte programının oldukça benzeridir.
- Burada da okuma yazma için bir int değişken kullanılmıştır.
- Farklı olarak okuma yazma aşamalarında int değişkenimiz son 16 bitlik kısmında karakter eşdeğeri saklayarak kopyalama yapar. (CopyByte da ise int değişken son kısmında 8bit lik veri saklamaktadır.)
- Byte Streamleri Kullanan Karakter Streamleri
  - Karakter streamleri I/O işlemleri gerçekleyebilmek için byte streamleri örtü gibi sararak da kullanılabilirler.
  - Örneğin FileReader FileInputStream i kullanması gibi.
- Genel amaçlı iki adet byte-to-character köprüsi olan akış InputStreamReader ve OutputStreamWriter bulunmaktadır.
- Network programlamada bu akışlar detaylıca kullanılacaktır.

# Satır Tabanlı I/O

---

- Karakter I/O işlemleri tekil karakter yerine genelde satır tabanlı olarak gerçekleştirilir.
- Bir satırın sonlanması için carriage-return/line-feed ("\r\n") kullanılır.
- Önceki örneği BufferedReader ve PrintWriter kullanarak genişletelim.

# Satır Tabanlı I/O

---

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

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

        BufferedReader inputStream = null;
        PrintWriter outputStream = null;

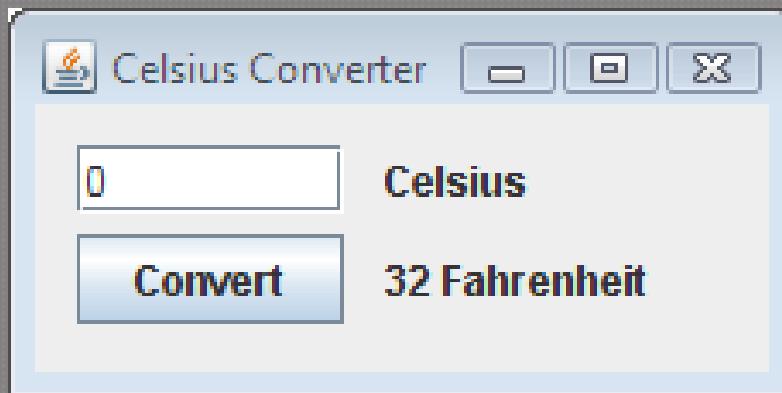
        try {
            inputStream = new BufferedReader(new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

# Swing ile Grafik Programlama

---

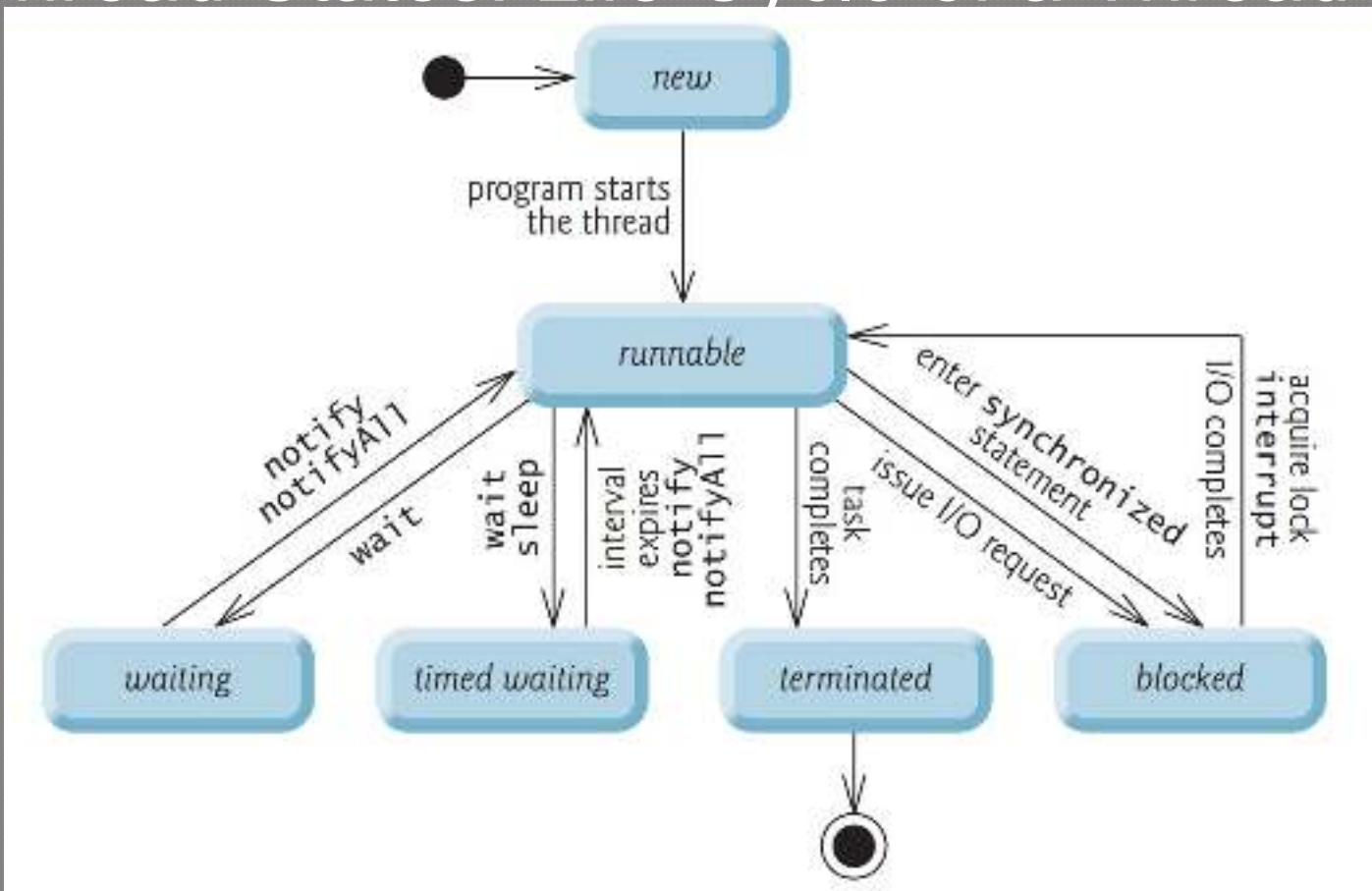
- HelloWorld Swing.java (örnek uygulama)
- Derece Çeviri Uygulaması
- CelsiusConverterGUI.java



- <http://docs.oracle.com/javase/tutorial/uiswing/learn/settingup.html>

# Multi-Thread Uygulama Geliştirme

- Thread States: Life Cycle of a Thread



# Thread Yaşam Döngüsü

---

- New ve Runnable State:
  - Bir thread yaşam döngüsüne ‘new’ durumunda başlar.
  - Program içerisinde ilgili thread çalıştırılana kadar (‘Runnable’ a geçene kadar) ‘new’ durumunda bekler.
  - ‘Runnable’ durumundaki bir thread ilgili işlemi yapıyor manasındadır.
- Waiting State:
  - Başka bir thread’ın çalışmaya devam etmesi durumunda ilgili thread’ın beklemeye geçmesidir.
- Timed Waiting State:
  - Bellirli bir zaman boyunca bekleme konumuna geçmesidir. İlgili zaman dilimi dolduktan sonra (expire) tekrar runnable state’e geçilir.
  - Başka bir threadin çalışmasından dolayı bir thread timed waiting durumuna da geçirilebilir. Tekrar aktif hale gelmesi diğer thread tarafından uyarılması veya expire olmayla olur.
  - Sleep moduna alınan threadlerde sleep interval boyunca timed waiting state de bulunurlar.

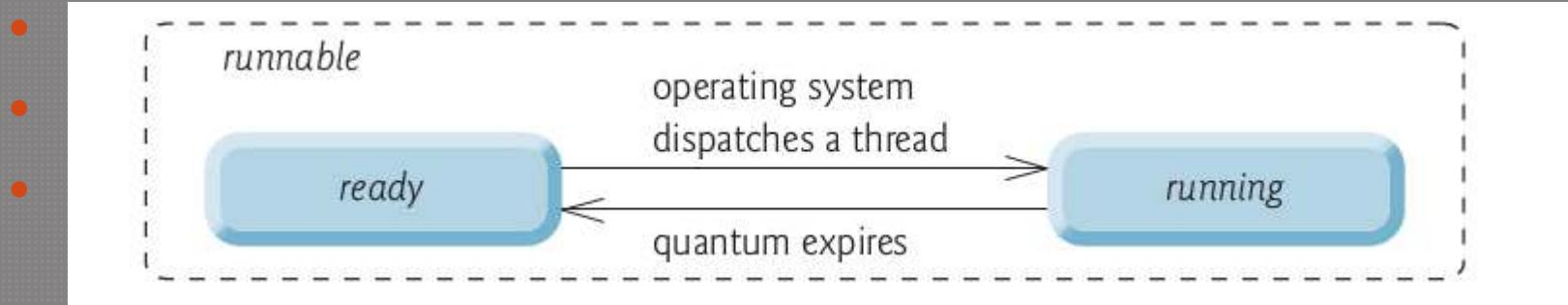
# Thread Yaşam Döngüsü

---

- Blocked State:
  - Bir işlemi anında gerçekleştiremeyen threadler blocked durumuna düşer ve geçici olarak işlemin tamamlanmasını bekler. (I/O erişimleri)
  - Blocked bir thread işlemciler boşta olsa da hi kullanamazlar.
- Terminated State:
  - İşlemlerini tamamlamış ve kapanan thread için geçerli durumdur. (dead state olarakda söylenir.)

# İşletim Sistemi açısından Runnable State

- İşletim sistemi için runnable state iki parçadan oluşur. (ready & running) JVM bunu tek state olarak görür.
- Ready : new durumundan runnable'a geçişdeki durumdur.
- Running: ready durumundaki bir thread'e işletim sisteminin bir işlemci atadığı andaki durumdur. (detayı İşletim Sistemleri dersi konusudur.)



```
import java.util.Random;

public class PrintTask implements Runnable
{
    private final int sleepTime; // random sleep time for thread
    private final String taskName; // name of task
    private final static Random generator = new Random();

    // constructor
    public PrintTask( String name )
    {
        taskName = name; // set task name

        // pick random sleep time between 0 and 5 seconds
        sleepTime = generator.nextInt( 5000 ); // milliseconds
    } // end PrintTask constructor

    // method run contains the code that a thread will execute
    public void run()
    {
        try // put thread to sleep for sleepTime amount of time
        {
            System.out.printf( "%s going to sleep for %d milliseconds.\n",
                taskName, sleepTime );
            Thread.sleep( sleepTime ); // put thread to sleep
        } // end try
        catch ( InterruptedException exception )
        {
            System.out.printf( "%s %s\n", taskName,
                "terminated prematurely due to interruption" );
        } // end catch

        // print task name
        System.out.printf( "%s done sleeping\n", taskName );
    } // end method run
} // end class PrintTask
```

```
1 // Fig. 26.4: TaskExecutor.java
2 // Using an ExecutorService to execute Runnables.
3 import java.util.concurrent.Executors;
4 import java.util.concurrent.ExecutorService;
5
6 public class TaskExecutor
7 {
8     public static void main( String[] args )
9     {
10         // create and name each runnable
11         PrintTask task1 = new PrintTask( "task1" );
12         PrintTask task2 = new PrintTask( "task2" );
13         PrintTask task3 = new PrintTask( "task3" );
14
15         System.out.println( "Starting Executor" );
16
17         // create ExecutorService to manage threads
18         ExecutorService threadExecutor = Executors.newCachedThreadPool();
19
20         // start threads and place in runnable state
21         threadExecutor.execute( task1 ); // start task1
22         threadExecutor.execute( task2 ); // start task2
23         threadExecutor.execute( task3 ); // start task3
24
25         // shut down worker threads when their tasks complete
26         threadExecutor.shutdown();
27
28         System.out.println( "Tasks started, main ends.\n" );
29     } // end main
30 } // end class TaskExecutor
```

