

## Laboration 16 – Exception, javadoc och jar

Avsikten med laborationen är att du ska träna på att använda exceptions och skriva några klassmetoder. Det är också några övningar om dokumentation och att skapa jar-filer.

### Grundläggande uppgifter

#### Uppgift 1a

Nedanstående rader med kod låter användaren mata in ett heltal. Om användaren matar in något som inte kan göras om till ett heltal så kastas ett *NumberFormatException* vilket fångas upp och hanteras (användaren får göra inmatning på nytt). När användaren matar in ett korrekt värde skrivs det ut.

Placera raderna med kod (nedan) i en main-metod och testkör dem. Använd sedan debuggern och gå framåt med *Step over*. Test med olika inmatningar och se hur de hanteras i programmet (när tilldelas inmatningOK värdet true?).

```
int value = 0;
boolean inputOK = false;
do {
    try {
        value = Integer.parseInt(JOptionPane.showInputDialog("Mata in ett heltal"));
        inputOK = true;
    }
    catch( NumberFormatException ex ) {}
} while( inputOK == false );
System.out.println("Inmatat tal: " + value);
```

#### Uppgift 1b

Skapa klassen *InOut* i paketet *resources*.

Skriv metoden *readInt()* i klassen *InOut*.

```
public int readInt()
```

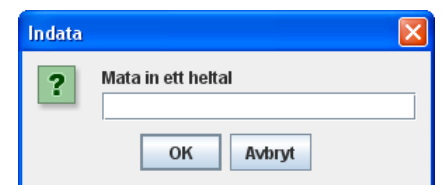
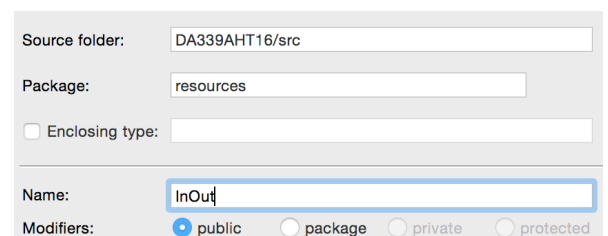
vilken ska låta användaren mata in ett heltal (typen *int*).  
Om användaren matar in något som inte kan göras om till en *int* ska användaren mata in ett nytt värde.

Du har stommen till lösningen i uppgift 1a. Men utskriftraden i slutet måste ersättas med en return-sats.

Testa din lösning med följande program:

```
package laboration16;
import resources.InOut;

public class Exerciselb {
    public static void main( String[] args ) {
        InOut io = new InOut();
        int nbr = io.readInt();
        System.out.println( "Inmatat tal: " + nbr );
    }
}
```



## Uppgift 1c

Skriv metoden ***readInt( String )*** i klassen *InOut*.

```
public int readInt( String txt )
```

Metoden ska låta användaren mata in ett heltal (typen *int*). Strängen *txt* ska visas i inmatningsdialogen. T.ex. ska anropet:

```
InOut io = new InOut();  
int nbr = io.readInt( "Mata in ett positivt heltal" );
```

visa inmatningsdialogen till höger.

Om användaren matar in något som inte kan göras om till en *int* ska användaren mata in ett nytt värde.

Testa din lösning med följande program:

```
package laboration16;  
import resources.InOut;  
  
public class Exerciselc {  
    public static void main( String[] args ) {  
        InOut io = new InOut();  
        int nbr = io.readInt( "Mata in ett tal utan decimaler" );  
        System.out.println( "Inmatat tal: " + nbr );  
    }  
}
```



## Uppgift 1d

Skriv metoderna ***readDouble( String txt )*** och ***readDouble()***. Metoderna ska placeras i klassen *InOut*. Båda metoderna ska låta användaren mata in ett decimaltal. Skillnaden är samma som i uppgift 1b och uppgift 1c.

En skillnad mot inläsningen av int-värde är att metoden *Double.parseDouble* även kan kasta *NullPointerException* (ej enligt dokumentationen men i vart fall i version 1.6 av java). Det innebär att både *NumberFormatException* och *NullPointerException* måste fångas:

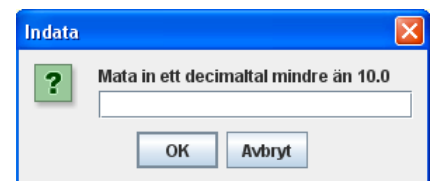
```
try {  
    :  
}  
catch( NumberFormatException ex1 ) {}  
catch( NullPointerException ex2 ) {}
```

### 1. Skriv metoden

```
public double readDouble( String )
```

Din lösning kan likna lösningen i uppgift 1c.

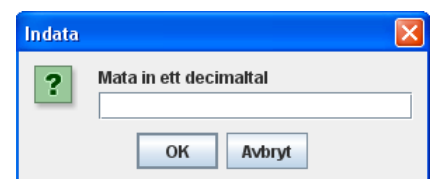
```
double tal = io.readDouble("Mata in ett decimaltal mindre än 10.0");
```



### 2. Skriv metoden

```
public double readDouble()
```

Vid inmatningen ska texten "Mata in ett decimaltal" visas. I din lösning ska du anropa metoden `readDouble( String )`.



Du ska testa dina lösningar med program liknande dem i uppgift 1c och uppgift 1b.

## Uppgift 2

Samtliga metoder du skrivit i klassen **InOut** är oberoende av instansvariabler (finns inte några i klassen). Sådana metoder kan och bör skrivas som klassmetoder, dvs. med modifieraren **static**.

Lägg till modifieraren **static** i samtliga metoder i klassen **InOut**. Kompilera klassen och testkör sedan metoderna med programmet:

```
package laboration16;
import resources.InOut;

public class Exercise2 {
    public static void main( String[] args ) {
        double real;
        int nbr;
        real = InOut.readDouble();
        System.out.println( "Inmatat tal: " + real );
        real = InOut.readDouble( "Mata in ett stort decimaltal" );
        System.out.println( "Inmatat tal: " + real );
        nbr = InOut.readInt();
        System.out.println( "Inmatat tal: " + nbr );
        nbr = InOut.readInt("Mata in ett positivt heltal");
        System.out.println( "Inmatat tal: " + nbr );
    }
}
```

## Uppgift 3

Uppgift 3 går ut på att skapa javadoc-dokumentation från Eclipse. Hämta filen **Commodity.java** från kurssidan och placera filen i paketet **laboration16**.

### Skriva javadoc-kommentarer i klassen Commodity

I klassen **Commodity** ska du skriva in javadoc-kommentarer. När du kommenterat färdigt ska du framställa javadoc-dokument.

En javadoc-kommentar inleds med **/\*\*** och avslutas med **\*/**. Varje rad ska börja med en **\***. T.ex.:

```
/**
 * Klassen Commodity representerar en vara i en butik.
 * @version 1.0
 * @author Rolf Axelsson
 */
```

Börja med att skriva in ovanstående kommentar före klassdeklarationen, dvs. ovanför raden `public class Commodity {`

Men ändra till ditt eget namn efter **@author**.

Varje **public konstruktor** och **metod** ska javadoc-kommenteras avseende funktion, parametrar och returvärde.

Du ska börja med att kommentera **konstruktorerna**. Skriv nedanstående kommentar precis ovanför konstruktorn med tom parameterlista:

```
/**
 * Konstruerar och initialiserar en vara med name="", category="",
 * quantity=0 och price=0.0.
 */
```

Skriv nedanstående kommentar precis ovanför konstruktorn med parameterlistan ( `String name, String category, int quantity, double price` ). Varje parameter kommenteras speciellt med

en `@param`-tagg.

```
/**
 * Konstruerar och initialiserar en vara med angivna värden.
 * @param name varans namn
 * @param category varans kategori
 * @param quantity antal varor på lagret
 * @param price varans pris
 */
```

Nu är det dags att kommentera *metoderna* i klassen. Börja med att lägga till nedanstående kommentar precis ovanför *getName*-metoden. Eftersom metoden har ett returvärde (String) så ska returvärdet kommenteras med `@return`-tagg.

```
/**
 * Returnerar varans namn.
 * @return varans namn
 */
```

Kommentera nu övriga *get-metoder* på motsvarande sätt. När du är färdig så fortsätt med nästa mening.

Skriv nedanstående kommentar direkt ovanför metoden *setName*.

```
/**
 * Sätter namnet på varan.
 * @param name namnet på varan
 */
```

Kommentera nu övriga *set-metoder* på motsvarande sätt.

Skriv nedanstående kommentar precis ovanför *changeQuantity*-metoden. Eftersom metoden har parameter och returvärde kommer kommentaren innehålla `@param`-tagg och `@return`-tagg.

```
/**
 * Ändrar antalet varor i lager. Ett positivt värde ökar antalet varor (inköp)
 * och ett negativt värde minskar antalet varor (försäljning)
 * @param change förändring av antalet varor
 * @return antal varor i lager efter förändring
 */
```

## Framställa javadoc-dokument med Eclipse

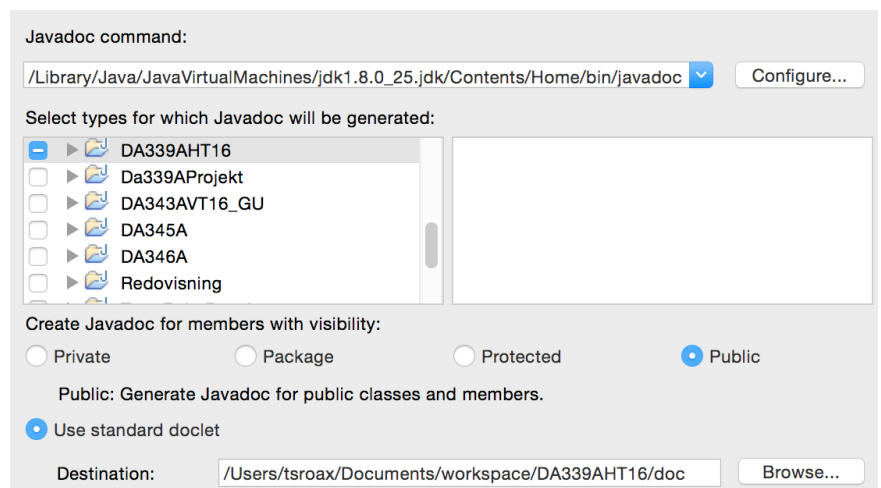
Det är enkelt att generera javadoc-dokumentation från Eclipse.

Högerklicka filen *Commodity.java* och välj

*Export... - Javadoc*.

I dialogen som visar sig måste du eventuellt ange var programmet *javadoc.exe* finns (under *Javadoc command*). Klicka i så fall på *Configure* och leta dig fram till *javadoc.exe*.

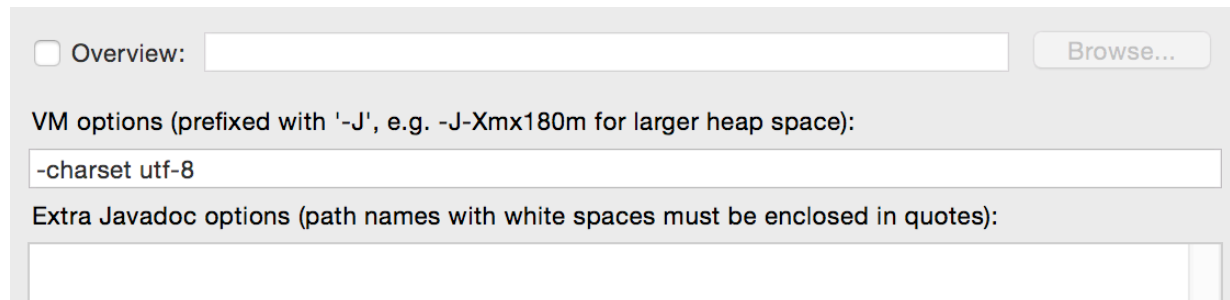
Klicka sedan på *Next* 2 gånger.



Nu blir dialogen till höger synlig. Om du använder å, ä, ö (eller andra bokstäver utanför a-z) så måste du ange det charset du använder.

Skriv *-charset utf-8* under *VM options*.

Klicka sedan på *Finish*.



Öppna doc – laboration16 i projektfönstret (nedanför src) och dubbelklicka *Commodity.html*. Om *javadoc.exe* är version 1.6 eller tidigare ser dokumentet lite annorlunda ut.

laboration16

## Class Commodity

java.lang.Object  
laboration16.Commodity

```
public class Commodity  
extends java.lang.Object
```

Klassen Commodity representerar en vara i en butik.

Version:

1.0

Author:

Rolf Axelsson

### Constructor Summary

#### Constructors

##### Constructor and Description

**Commodity**()

Konstruerar och initialiserar en vara med name="", category="", quantity=0 och price=0.0.

**Commodity**(java.lang.String name, java.lang.String category, int quantity, double price)

Konstruerar och initialiserar en vara med angivna värden.

## Uppgift 4

En **JAR-fil** (arkiv-fil), Java ARchive file, kan innehålla alla de filer som behövs i ett program. JAR-filen kan dessutom göras körbar (Uppgift 5-7). Exekveringen av programmet startar då användaren dubbelklickar jar-filen.

Om du vill använda bilder och ljud i programmet så går det bra (Uppgift 6).

**Uppgift 4** går ut på att skapa en jar-fil. jar-filen ska innehålla paketet *resources*. Paketet ska bl.a. innehålla klassen *Stopwatch*, vilken kommer att användas av klassen *EstimateTime*.

## Göra en jar-fil

- Skapa ett nytt projekt som heter **CreateJAR**.
- Markera paketet *resources* (som du nyss arbetat med), kopiera det och klistra in det i *src*-katalogen i det nya projektet.
- Klistra in *Stopwatch.java* (från föreläsning 3) i paketet *resources*. Glöm inte ändra till `package resources;` i början av filen. Spara filen efter ändringen. Det ska inte vara några rödmarkeringar i filen efter du sparat den.
- Högerklicka paketet *resources* (i Eclipse) och välj Export...
- Markera *Java – JAR File* och klicka sedan på *Next >*



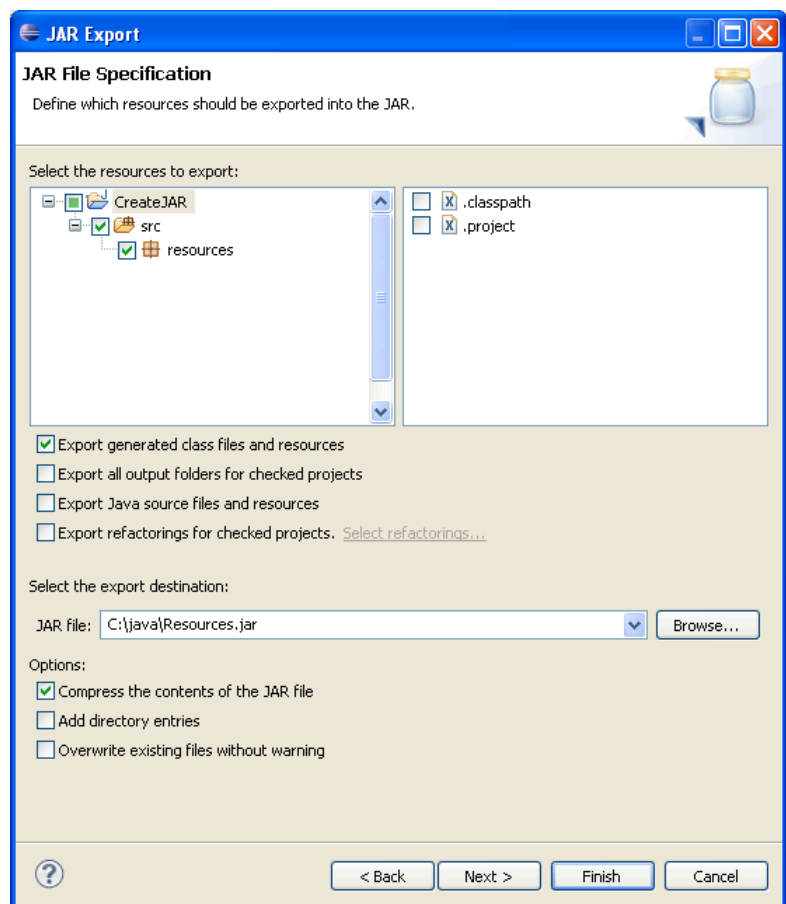
- Nu kan du välja vilka paket som ska ingå i jar-filen genom att klicka på +-tecknet framför CreateJAR och src. Om projektet innehåller många paket så kan du här markera de paket som ska ingå.

Mitt i dialogrutan kan du välja vilka typer av filer som ska ingå. Just nu nöjer vi oss med class-filer och därför ska endast det översta alternativet vara markerat.

Klicka på **Browse** och välj den katalog i vilken jar-filen ska sparas. Skriv in **Resources.jar** efter filnamn och klicka på **Spara**.

Klicka på **Finish**.

- Nu kan du med hjälp av *Utforskaren* se att *Resources.jar* har skapats i vald katalog.



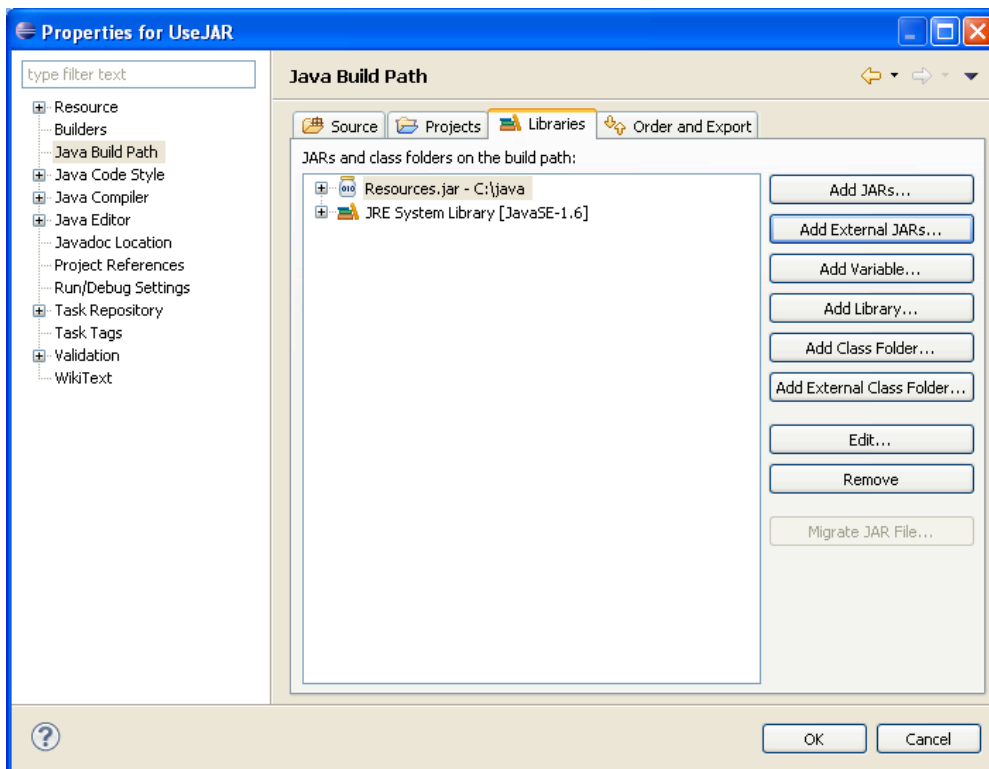
## Testa jar-filen i ett annat projekt

Nu ska innehållet i *Resources.jar* användas av ett program i ett helt nytt projekt. Skapa projektet *UseJAR*.

### Göra paketet *resources* tillgängligt i projekt

I projektet ska *Resources.jar* kunna användas. För detta krävs några inställningar:

1. Högerklicka projektet (i projektfönstret) och välj **Properties**. Markera **Java Build Path** och klicka på fliken **Libraries**. Klicka sedan på **Add External JARs...**.
2. Bläddra fram till *Resources.jar* och klicka sedan på **Öppna**. Nu ska dialogen se ut så här:



3. Klicka på **OK**. Nu kan innehållet i *Resources.jar* användas av program i projektet *UseJAR*.

### Använda klassen *Stopwatch* i arkivet *Resources.jar*

*EstimateTime* (från föreläsning 3) använder klassen *Stopwatch*. Skapa paketet *f3* i *UseJAR* och kopiera sedan klassen *EstimateTime* till *f3*.

Det blir rödmarkering i klassen *EstimateTime* där *Stopwatch*-objekt skapas. Det beror på att det inte finns någon klass med namnet *Stopwatch* i paketet *f6*. Men klassen *Stopwatch* finns i paketet *resources* i arkivfilen *Resources.jar*. Och innehållet i *Resources.jar* är tillgängligt efter punkterna 1 och 2 ovan. Lägg till en import-sats efter package-kommandot i *EstimateTime*:

```
package f3;  
import resources.*;
```

När du lagt till ovanstående 2 rader i *EstimateTime.java* är det dags att köra programmet.

## Uppgift 5

Uppgift 5 går ut på att göra en körbar jar-fil, dvs. ett program som exekveras vid dubbelklick på programikonen. För att en jar-fil ska vara körbar krävs det att en klass i jar-filen innehåller en main-metod.

Skapa projektet **RunnableJar**. I projektet ska du lägga till paketet **hello**. Klasserna *HelloWorld* och *HelloMain* ska placeras i paketet *hello*.

```
package hello;
import javax.swing.*;

public class HelloWorld {
    private String hello;

    public HelloWorld() {
        this.hello = "Hello, world";
    }

    public HelloWorld( String anotherHello ) {
        this.hello = anotherHello;
    }

    public void sayHello() {
        JOptionPane.showMessageDialog( null, this.hello );
    }
}

-----
package hello;

public class HelloMain {
    public static void main( String[] args ) {
        HelloWorld hw = new HelloWorld( "Hej, världen" );
        hw.sayHello();
    }
}
```

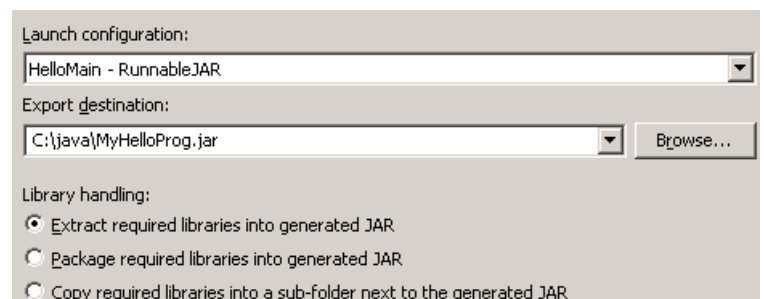


Testkör *HelloMain* så att programmet fungerar. Ett fönster liknande fönstret ovan ska visa sig.

### Skapa en körbar jar-fil

Nu ska du skapa en jar-fil på liknande sätt som i Uppgift 4. Men välj **Export – java – Runnable JAR file**. I dialogen ska du:

- ange klassen som innehåller main-metoden (Launch configuration). Välj *HelloMain – RunnableJAR* ur listan
- vad jarfilen ska heta och var filen ska placeras på hårddisken (Export destination)



Klicka sedan på **Finish**.

Nu ska du dubbelklicka *MyHelloProg.jar* för att kontrollera att den är körbar.

Om du testar **MyHelloProg.jar** på en annan dator kommer den endast fungera om JRE (Java Runtime Environment) är installerat på datorn.



## Extrauppgifter

### Uppgift 6

I uppgift 6 ska en bild visas tillsammans med texten "Hej världen". Ändra klassen *HelloWorld* så att den ser ut så här:

```
package hello;
import java.applet.AudioClip;
import javax.swing.*; // ImageIcon, JApplet, JLabel, JOptionPane

public class HelloWorld {
    private String hello;

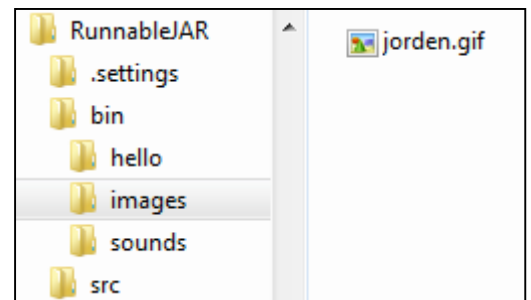
    public HelloWorld() {
        this.hello = "Hello, world";
    }

    public HelloWorld( String anotherHello ) {
        this.hello = anotherHello;
    }

    public void sayHello() {
        ImageIcon image = new ImageIcon(getClass().getResource (
"/images/jorden.gif" ));
        AudioClip clip = JApplet.newAudioClip(getClass().getResource (
"/sounds/flipper.wav"));
        clip.play();
        JLabel message = new JLabel(this.hello);
        message.setIcon(image);
        JOptionPane.showMessageDialog( null, message );
    }
}
```

Skapa en katalog **images** i projekt-katalogen **bin**. I katalogen **images** ska du placera bildfilen **jorden.gif**.

Skapa på samma sätt katalogen **sounds** i projekt-katalogen **bin**. Placera ljudfilen **flipper.wav** i katalogen.



Kompilera *HelloWorld* och testkör *HelloMain*. Nu ska en dialog liknande figuren nedan visa sig, och samtidigt ska ljudfilen **flipper.wav** spelas.



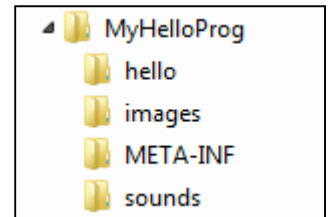
Om du fick upp ovanstående dialog och ljudet spelades upp så ska du skapa en körbar jar-fil på nytt.

Gör på samma sätt som i Uppgift 5. Tyvärr så måste bild och ljudfiler läggas till för hand. Gör så här:

Leta upp ***MyHelloProg.jar*** som du precis tillverkat. Packa upp innehållet till en mapp. I figuren till höger heter mappen MyHelloProg. Eventuellt måste du byta filtypen till .zip för att kunna packa upp innehållet. Kopiera katalogerna ***images*** och ***sounds*** till mappen.

Skapa nu en ny zip-fil som innehåller katalogerna hello, images, META-INF och sounds. Du kan ge zip-filen namnet MyHelloProg.zip. Ändra slutligen filändelsen till .jar (släng den gamla MyHelloProg.jar först).

Testkör nu jar-filen!



## Uppgift 7

Gör en körbar JAR-fil av ditt Sten, sax och påse – spel.

## Uppgift 8

Kör och studera animation-exemplen i föreläsningsunderlaget. Testa ideer, ändra i klasser och kör programmen på nytt.

## Förslag till lösningar på laboration 16

### Uppgift 1b

```
package resources;
import javax.swing.JOptionPane;

public class InOut {

    public int readInt() {
        int value = 0;
        boolean inputOK = false;
        do {
            try {
                value = Integer.parseInt(JOptionPane.showInputDialog("Mata in ett heltal"));
                inputOK = true;
            } catch (NumberFormatException ex) {}
        } while (inputOK == false);
        return value;
    }
}
```

### Uppgift 1c

```
public int readInt( String txt ) {
    int value = 0;
    boolean inputOK = false;
    do {
        try {
            value = Integer.parseInt(JOptionPane.showInputDialog( txt ) );
            inputOK = true;
        }
        catch( NumberFormatException ex ) {}
    } while( inputOK == false );
    return value;
}
```

### Uppgift 1d

```
public static double readDouble( String txt ) {
    double value = 0;
    boolean inputOK = false;
    do {
        try {
            value = Double.parseDouble(JOptionPane.showInputDialog( txt ) );
            inputOK = true;
        }
        catch( NumberFormatException ex1 ) {}
        catch( NullPointerException ex2 ) {}
    } while( inputOK == false );
    return value;
}

public static double readDouble() {
    return readDouble( "Mata in ett decimaltal" );
}
```

## Uppgift 2

```
package resources;
import javax.swing.JOptionPane;

public class InOut {
    public static int readInt() {
        int value = 0;
        boolean inputOK = false;
        do {
            try {
                value = Integer.parseInt(JOptionPane.showInputDialog("Mata in ett
heltal"));
                inputOK = true;
            } catch (NumberFormatException ex) {
            }
        } while (inputOK == false);
        return value;
    }

    public static int readInt( String txt ) {
        int value = 0;
        boolean inputOK = false;
        do {
            try {
                value = Integer.parseInt(JOptionPane.showInputDialog( txt ) );
                inputOK = true;
            }
            catch( NumberFormatException ex ) {}
        } while( inputOK == false );
        return value;
    }

    public static double readDouble( String txt ) {
        double value = 0;
        boolean inputOK = false;
        do {
            try {
                value = Double.parseDouble(JOptionPane.showInputDialog( txt ) );
                inputOK = true;
            }
            catch( NumberFormatException ex1 ) {}
            catch( NullPointerException ex2 ) {}
        } while( inputOK == false );
        return value;
    }

    public static double readDouble() {
        return readDouble( "Mata in ett decimaltal" );
    }
}
```

### Uppgift3

```
package laboration16;

/**
 * Klassen Commodity representerar en vara i en butik.
 * @version 1.0
 * @author Rolf Axelsson
 */
public class Commodity {
    private String name;
    private String category;
    private int quantity;
    private double price;

    /**
     * Konstruerar och initialiserar en vara med name="", category="",
     * quantity=0 och price=0.0.
     */
    public Commodity() {
        this.name = "";
        this.category = "";
    }

    /**
     * Konstruerar och initialiserar en vara med angivna värden.
     * @param name varans namn
     * @param category varans kategori
     * @param quantity antal varor på lagret
     * @param price varans pris
     */
    public Commodity(String name, String category, int quantity, double price) {
        this.name = name;
        this.category = category;
        this.quantity = quantity;
        this.price = price;
    }

    /**
     * Returnerar varans namn.
     * @return varans namn
     */
    public String getName() {
        return this.name;
    }

    /**
     * Sätter varans namn
     * @param name varans namn
     */
    public void setName(String name) {
        this.name = name;
    }

    /**
     * Returnerar varans kategori.
     * @return varans kategori
     */
    public String getCategory() {
        return this.category;
    }

    /**
     * Sätter varans kategori
     * @param category varans kategori
     */
    public void setCategory(String category) {
        this.category = category;
    }
}
```

```
/**
 * Returnerar varans pris.
 * @return varans pris
 */
public double getPrice() {
    return this.price;
}

/**
 * Sätter varans pris
 * @param price varans pris
 */
public void setPrice(double price) {
    this.price = price;
}

/**
 * Ändrar antalet varor i lager. Ett positivt värde ökar antalet varor (inköp)
 * och ett negativt värde minskar antalet varor (försäljning)
 * @param change förändring av antalet varor
 * @return antal varor i lager efter förändring
 */
public int changeQuantity(int change) {
    quantity += change;
    return this.quantity;
}

/**
 * Skriver ut information om ett objekt
 */
public void info() {
    System.out.println(this.name + ", " + this.category + "\n" +
        "Antal i lager: " + this.quantity + "\n" +
        "Pris: " + this.price + " kr");
}
}
```