# Programs and JDK-Documentation

Examination in Object Oriented Programming W 2017

University of Applied Science Ravensburg-Weingarten
Prof. Dr. M. Zeller

**Please do not write on these sheets.**

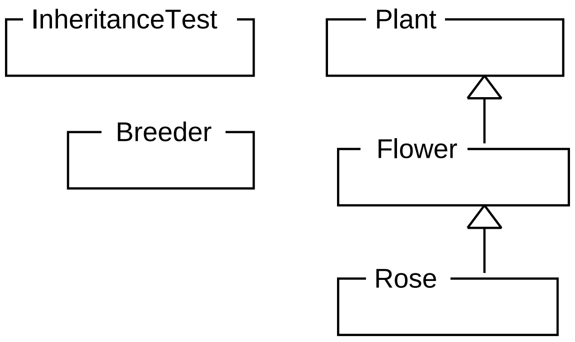
## Contents

## 1 Program Exception

```
class BlueException extends Exception {
}

class GreenException extends BlueException {
}

class RedException extends BlueException {
}

public class CatchAndPrint {

    public static void main(String[] args) {
        CatchAndPrint cap = new CatchAndPrint();
        cap.bar();
    }

    public void foo(int num) throws Exception {
        System.out.println("num: " + num);
        if (num == 0) {
            throw new BlueException();
        }
        if (num == 1) {
            throw new RedException();
        }
        if (num == 2) {
            int res = 10 / (num - 2);
        }
        if (num == 3) {
            throw new GreenException();
        }
        if (num == 4) {
            int[] numbers = new int[4];
            numbers[num] = num;
        }
    }

    public void bar() {
        for (int i = 0; i < 5; i++) {
            try {
                foo(i);
            } catch

          ⋮

        }
    }
}
```

## 2 Program Constructors

The class `Breeder` is defined in the package `persons`. All other classes of this program are defined in the package `exoop`.



```
package exoop;
import persons.Breeder;
public class InheritanceTest {
    void test() {
        System.out.println("———— 1 ————");
        Plant aPlant = new Plant();
        System.out.println("———— 2 ————");
        aPlant.grow();
        System.out.println("———— 3 ————");
        Flower aFlower = new Flower();
        System.out.println("———— 4 ————");
        aFlower.grow();
        System.out.println("———— 5 ————");
        aFlower.bloom();
        System.out.println("———— 6 ————");
        Rose aRose = new Rose();
        System.out.println("———— 7 ————");
        aRose.grow();
        System.out.println("———— 8 ————");
        aRose.bloom();
        System.out.println("———— 9 ————");
        Breeder aBreeder = new Breeder();
        System.out.println("———— 10 ————");
        Plant rPlant = new Rose();
        System.out.println("———— 11 ————");
        // rPlant.grow();
        // rPlant.bloom();
        // aBreeder.name = "Evers";
        // aBreeder.country = "France";
    }
}

class Plant {
    String id = "general plant";

    Plant() {
        System.out.println(" creating a plant: ");
    }

    void grow() {
        System.out.println(" growing plant: " + id);
    }
}
```

```
class Flower extends Plant {

    Breeder breeder = new Breeder("Dicksons");

    void bloom() {
        grow();
        System.out.println(" flower is blooming: " + id);
    }
}

class Rose extends Flower {

    Rose() {
        id = "Nice Rose";
        System.out.println(" creating a rose: " + id);
    }

    void grow() {
        System.out.println(id + " is growing ");
        System.out.println(" Breeder is: " + breeder.name);
    }
}

package persons;
public class Breeder {
    public String name;
    String country = "Germany";

    public Breeder() {
        country = "UK";
        System.out.println(" creating a breeder");
    }

    public Breeder(String theName) {
        name = theName;
        System.out.println(" creating a breeder in: " + country);
    }
}
```

## 3 Program Network

```java
public class Network {
    Junktion start;

    public static void main(String[] args) {
        Network aNetwork = new Network();
        aNetwork.makeNet();
        aNetwork.printNetwork(aNetwork.start);
    }

    public void makeNet() {
        Junktion junk_1 = new Junktion(21);
        Junktion junk_2 = new Junktion(32);
        Junktion junk_3 = new Junktion(13);
        Junktion junk_4 = new Junktion(64);
        Junktion junk_5 = new Junktion(55);

        start = junk_3;
        junk_3.up = junk_1;
        junk_3.down = junk_5;
        start.up.down = junk_4;
        junk_1.up = junk_2;
    }

    void printNetwork(Junktion junk) {

          ⋮

    }
}

public class Junktion {
    int value;
    Junktion up;
    Junktion down;

    Junktion(int theValue) {
        value = theValue;
    }

    void print() {
        System.out.println("Junktion: " + value);
    }
}
```

# 4 Program Collections and IO

```
public class ObjectStorage {

    Customer[] customers = new Customer[4];
    . . . . . . . . . . .   custMap = . . . . . . . . . . . . . . . .

    public void runObjectStorage() {
        ObjectStorage oTest = new ObjectStorage();
        oTest.initCustomer();
        oTest.printCustomers();

        oTest.storeInHashMap();
        oTest.accessHashMap();

        oTest.writeCustomer("custFile.dat");
        oTest.readCustomer("custFile.dat");
    }

    void initCustomer() {
        Booking aBooking = new Booking("Bilton", 1111);
        Customer cust = new Customer(30111, (short) 1991, "Max");
        cust.theBooking = aBooking;
        customers[0] = cust;

        aBooking = new Booking("Eastern", 2222);
        cust = new Customer(30222, (short) 1992, "Mia");
        cust.theBooking = aBooking;
        customers[1] = cust;

        aBooking = new Booking("Western", 3333);
        cust = new Customer(30333, (short) 1993, "Finn");
        cust.theBooking = aBooking;
        customers[2] = cust;

        aBooking = new Booking("Astoria", 4444);
        cust = new Customer(30444, (short) 1994, "Emma");
        cust.theBooking = aBooking;
        customers[3] = cust;
    }

    void storeInHashMap() {
        for (Customer aCustomer : customers) {
            Booking key = . . . . . . . . . . . .
            . . . . . . . . . . . . . . . . . . . . .
        }
    }
    void accessHashMap() {
        System.out.println(" ----- accessing HashMap ----- ");
        Booking aBooking = new Booking("Eastern", 2222);
        Customer aCustomer = . . . . . . . . . . . . . . . . . . . .
        aCustomer.print();
        aCustomer = new Customer(30555, (short) 1995, "Tom");
        aCustomer.theBooking = aBooking;
         . . . . . . . . . . . . . . . . . . .
        aCustomer = custMap.get(aBooking);
        aCustomer.print();
        custMap.remove(aBooking);
   //  aCustomer = custMap.get(aBooking);
   // System.out.println("last access: " + aCustomer);
    }

    void writeCustomer(String fileName) {
        try (BufferedOutputStream bufs = getBufferedOutputStream(fileName);
                  . . . . . . .   datOS =  . . . . . . . . . . . . . ) {
            for (int i = 0; i < customers.length; i = i + 1) {
                              :
                              :
            }
        } catch . . . . . . . . . . . .
                             :
    }

    void readCustomer(String fileName) {
        try (BufferedInputStream bufs = getBufferedInputStream(fileName);
                  . . . . . . .  datIS . . . . . . . . . . . . . . . . ) {
            for (int i = 0; i < customers.length; i = i + 1) {
                              :
            }
        } catch . . . . . . . . . . . .
                             :
                             :
    }

    BufferedOutputStream getBufferedOutputStream(String fileName) throws IOException
        FileOutputStream fileOs = new FileOutputStream(fileName);
        BufferedOutputStream bufs = new BufferedOutputStream(fileOs);
        return bufs;
    }

    BufferedInputStream getBufferedInputStream(String fileName) throws IOException {
        FileInputStream fileIs = new FileInputStream(fileName);
        BufferedInputStream bufs = new BufferedInputStream(fileIs);
        return bufs;
    }

    void printCustomers() {
        for (Customer aCustomer : customers) {
            aCustomer.print();
        }
    }
}
```

```
class Customer {
    public int custId;
    short yearOfBirth;
    String name;
    Booking theBooking;

    Customer(int custId, short yearOfBirth_, String name_) {
        custId = custId;
        yearOfBirth = yearOfBirth_;
        name = name_;
    }

    void print() {
        System.out.print("Customer: . . . . .
                :
    }
}
class Booking {
    String hotel;
    int bookingId;
    boolean payed;

    Booking(String name_, int bookingId_) {
        hotel = name_;
        bookingId = bookingId_;
    }

    public int hashCode() {
        int hCode = hotel.hashCode();
        hCode = hCode ^ bookingId;
        hCode = hCode ^ (payed ? 1 : 0);
        return hCode;
    }

    public boolean equals(Object obj) {
        Booking otherBooking = (Booking) obj;
        if (!hotel.equals(otherBooking.hotel)) {
            return false;
        }
        if (bookingId != otherBooking.bookingId) {
            return false;
        }
        return payed == otherBooking.payed;
    }
}
```

# 5 Program Class Design

```
public class TestClass {
    int itemCount = 4;
    public void createItems() {
        ArrayList<Desk> deskList = new ArrayList<>();
        ArrayList<Printer> printerList = new ArrayList<>();
        ArrayList<TableLamp> lampList = new ArrayList<>();
        for (int i = 0; i < itemCount; i++) {
            Desk aDesk = new Desk(3 * i);
            Printer aPrinter = new Printer(3 * i + 1);
            TableLamp aLamp = new TableLamp(3 * i + 2);
            printerList.add(aPrinter);
            deskList.add(aDesk);
            lampList.add(aLamp);
        }
        for (Desk theDesk : deskList) {
            theDesk.print();
        }
        for (Printer thePrinter : printerList) {
            thePrinter.print();
        }
        for (TableLamp theLamp : lampList) {
            theLamp.print();
        }
    }
}
class Desk {
    short height = 75;
    int inventoryId;
    Desk(int ivId) {
        inventoryId = ivId;
    }
    void print() {
        System.out.println("desk: " + inventoryId + ", " + height);
    }
}
class Printer {
    int inventoryId;
    Printer(int ivId) {
        inventoryId = ivId;
    }
    void print() {
        System.out.println("printer: " + inventoryId);
    }
}
class TableLamp {
    int inventoryId;
    TableLamp(int ivId) {
        inventoryId = ivId;
    }
    void print() {
        System.out.println("table lamp: " + inventoryId);
    }
}
```

# 6 JDK Documentation

## 6.1 DataOutputStream Summary

```
public class DataOutputStream
```

### 6.1.1 Constructor

**DataOutputStream(OutputStream out)**
Creates a DataOutputStream that uses the specified underlying OutputStream.

### 6.1.2 Methods (Selection)

| | |
|---|---|
| void | flush()<br>Flushes this data output stream. |
| int | size()<br>Returns the current value of the counter written, the number of bytes written to this data output stream so far. |
| void | write(byte[] b, int off, int len)<br>Writes len bytes from the specified byte array starting at offset off to the underlying output stream. |
| void | write(int b)<br>Writes the specified byte (the low eight bits of the argument b) to the underlying output stream. |
| void | writeBoolean(boolean v)<br>Writes a boolean to the underlying output stream as a 1-byte value. |
| void | writeByte(int v)<br>Writes out a byte to the underlying output stream as a 1-byte value. |
| void | writeBytes(String s)<br>Writes out the string to the underlying output stream as a sequence of bytes. |
| void | writeChar(int v)<br>Writes a char to the underlying output stream as a 2-byte value, high byte first. |
| void | writeChars(String s)<br>Writes a string to the underlying output stream as a sequence of characters. |
| void | writeDouble(double v)<br>Converts the double argument to a long using the doubleToLongBits method in class Double, and then writes that long value to the underlying output stream as an 8-byte quantity, high byte first. |
| void | writeFloat(float v)<br>Converts the float argument to an int using the floatToIntBits method in class Float, and then writes that int value to the underlying output stream as a 4-byte quantity, high byte first. |
| void | writeInt(int v)<br>Writes an int to the underlying output stream as four bytes, high byte first. |
| void | writeLong(long v)<br>Writes a long to the underlying output stream as eight bytes, high byte first. |
| void | writeShort(int v)<br>Writes a short to the underlying output stream as two bytes, high byte first. |
| void | writeUTF(String str)<br>Writes a string to the underlying output stream using modified UTF-8 encoding in a machine-independent manner. |

## 6.2 DataInputStream Summary

`public class DataInputStream`

### 6.2.1 Constructor

**DataInputStream(InputStream in)**
Creates a DataInputStream that uses the specified underlying InputStream.

### 6.2.2 Methods (Selection)

| | |
|---|---|
| int | read(byte[] b)<br>Reads some number of bytes from the contained input stream and stores them into the buffer array b. |
| int | read(byte[] b, int off, int len)<br>Reads up to len bytes of data from the contained input stream into an array of bytes. |
| boolean | readBoolean()<br>Reads one input byte and returns true if that byte is nonzero, false if that byte is zero. |
| byte | readByte()<br>Reads and returns one input byte. |
| char | readChar()<br>Reads two input bytes and returns a char value. |
| double | readDouble()<br>Reads eight input bytes and returns a double value. |
| float | readFloat()<br>Reads four input bytes and returns a float value. |
| void | readFully(byte[] b)<br>Reads some bytes from an input stream and stores them into the buffer array b. |
| void | readFully(byte[] b, int off, int len)<br>Reads len bytes from an input stream. |
| int | readInt()<br>Reads four input bytes and returns an int value. |
| String | readLine()<br>Reads the next line of text from the input stream. |
| long | readLong()<br>Reads eight input bytes and returns a long value. |
| short | readShort()<br>Reads two input bytes and returns a short value. |
| int | readUnsignedByte()<br>Reads one input byte, zero-extends it to type int, and returns the result, which is therefore in the range 0 through 255. |
| int | readUnsignedShort()<br>Reads two input bytes and returns an int value in the range 0 through 65535. |
| String | readUTF()<br>See the general contract of the readUTF method of DataInput. |
| static String | readUTF(DataInput in)<br>Reads from the stream in a representation of a Unicode character string encoded in modified UTF-8 format; this string of characters is then returned as a String. |

## 6.3 HashMap Summary

### 6.3.1 Constructors (Selection)

**HashMap()**
Constructs an empty HashMap with the default initial capacity (16).

**HashMap(int initialCapacity)**
Constructs an empty HashMap with the specified initial capacity.

### 6.3.2 Methods (Selection)

| | |
|---|---|
| V | get(Object key)<br>Returns the value to which the specified key is mapped, or null if this map contains no mapping for the key. |
| V | put(K key, V value)<br>Associates the specified value with the specified key in this map. If the map previously contained a mapping for the key, the old value is replaced. |
| V | remove(Object key)<br>Removes the mapping for the specified key from this map if present. |
| int | size()<br>Returns the number of key-value mappings in this map. |
| Collection<V> | values()<br>Returns a Collection view of the values contained in this map. |