

## Exercise In chapter 1 page 20

Correct order:

```
class Shuffle1 {  
    public static void main(String [] args) {  
        int x = 3;  
        while (x > 0){  
            if (x > 2) {  
                System.out.print("a");  
                x = x - 1;  
                System.out.print("-");  
            }  
            if (x == 2) {  
                System.out.print("b c");  
                x = x - 1;  
                System.out.print("-");  
            }  
            if (x == 1) {  
                System.out.print("d");  
                x = x - 1;  
            }  
        }  
    }  
}
```

## Exercise in chapter 1 page 21

- A. it need a increment line in the while loop so that it can come to an end
- B. it needs a class decleration
- c. it need a void main method.

## Exercise in chapter 1 page 22

Down:

1 float

2.void

across:

4. Java

6. Loop

3. public

5. array

7 while

10 compiler

11 variable

13 it

15 static

16 main

17 method

20 jvm

8. branch

9. dc

12 int

13 ic

14 system.out.print

18 string

19 declare

21 command

### Exercise in chapter 1 page 23

Candidates:

`y = x - y;`

`y = y + x;`

`y = y + 2;  
if( y > 4 ) {  
 y = y - 1;  
}`

`x = x + 1;  
y = y + x;`

`if ( y < 5 ) {  
 x = x + 1;  
 if ( y < 3 ) {  
 x = x - 1;  
 }  
}`

Possible output:

22 46

11 34 59

02 14 26 38

02 14 36 48

00 11 21 32 42

11 21 32 42 53

00 11 23 36 410

02 14 25 36 47

### Exercise in chapter 1 page 24

```
class PoolPuzzleOne {
```

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

```
        int x = 0;
```

```
        while ( x < 4 ) {
```

```
            System.out.print("a");
```

```

if ( x < 1 ) {
    System.out.print(" ");
}
System.out.print("\n");
if ( x > 1) {
    System.out.print(" oyster ");
    x = x + 2;
}
if ( x == 1 ) {
System.out.print("noys ")
if ( x<1 ) {
System.out.print("oise ");
}
System.out.println();
X=x+1
}}}

```

### Exercise in chapter 2 page 42

- a. Song object declaration was not done
- b. Play() method does not exist

### Exercise in chapter 2 page 43

```

class DrumKit{
    boolean topHat = true;
    boolean snare = true;
    void playTopHat () {
        System.out.println("ding ding da-ding");
    }
    void playSnare() {
        System.out.println("bang bang ba-bang");
    }
}

```

```

class DrumKitTestDrive {
public static void main(String [] args) {
    DrumKit d = new DrumKit();
    if (d.snare == true) { d.playSnare(); }
    d.playTopHat();
}}

```

## Exercise in chapter 2 page 44

```

public class EchoTestDrive {
    public static void main(String[] args) {
        Echo e1 = new Echo();
        Echo e2 = new Echo( );
        While (x<4){
            e1.count = e1.count + 1;
            if ( x == 3) {
                e2.count = e2.count + 1; }
            if ( x>0 ) {
                e2.count = e2.count + e1.count; }
            x = x + 1;
        }
        System.out.println(e2.count);
    } }

```

## Exercise in chapter 2 page 45

My instance variable values can be different from my buddy's values. – object

I behave like a template. -class

I like to do stuff. – Method

I can have many methods.-class

I represent “state.” – instance variable

I have behaviors. – object

I am located in objects. Instance variable

I live on the heap. – object

I am used to create object instances. – class

My state can change. – object

I declare methods. – class

I can change at runtime. – instance variables

### **Exercise in chapter 3 page 63**

A the objects aren't made after declaring the array.

B. z increment can be done in the end and the loop condition could be  $z < 3$

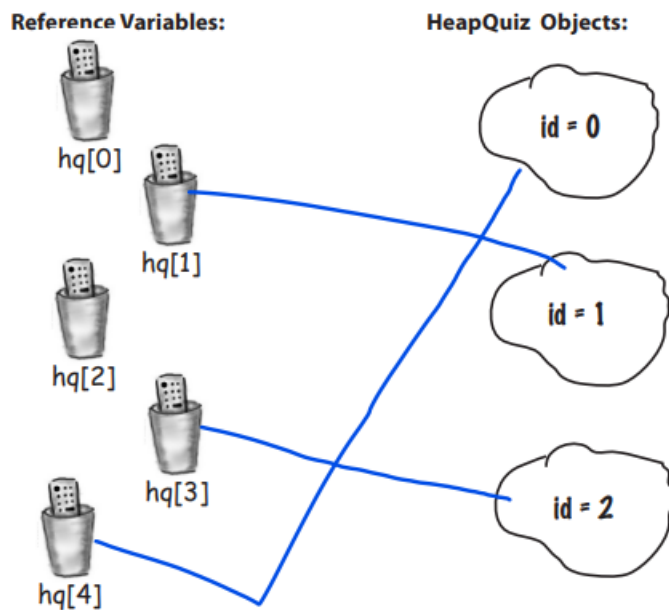
### **Exercise in chapter 3 page 64**

```
class TestArrays {  
    public static void main(String [] args){  
        int [] index = new int[4];  
        index[0] = 1;  
        index[1] = 3;  
        index[2] = 0;  
        index[3] = 2;  
        String [] islands = new String[4];  
        islands[0] = "Bermuda";  
        islands[1] = "Fiji";  
        islands[2] = "Azores";  
        islands[3] = "Cozumel";  
        int y = 0;  
        int ref;  
        while (y < 4) {  
            ref = index[y];  
            System.out.print("island = ");  
            System.out.println(islands[ref]);  
            y = y + 1;}  
    }  
}
```

## Exercise in chapter 3 page 65

```
class Triangle {
    double area;
    int height;
    int length;
    public static void main(String[] args) {
        int x = 0;
        Triangle[] ta = new Triangle[4];
        while (x < 4) {
            ta[x] = new Triangle();
            ta[x].height = (x + 1) * 2;
            ta[x].length = x + 4;
            ta[x].setArea();
            System.out.print("triangle " + x +
                ", area");
            System.out.println(" = " + ta[x].area);
            x = x + 1;
        }
        int y = x;
        x = 27;
        Triangle t5 = ta[2];
        ta[2].area = 343;
        System.out.print("y = " + y);
        System.out.println(", t5 area = " +
            t5.area);
    }
    void setArea() {
        area = (height * length) / 2;
    }
}
```

### Exercise in chapter 3 page 66



### Exercise in chapter 4 page 87

```
int a = calcArea(7, 12); ✓  
short c = 7;  
calcArea(c, 15); ✗  
  
int d = calcArea(57); ✗  
  
calcArea(2, 3); ✓  
  
long t = 42;  
int f = calcArea(t, 17); ✗  
  
int g = calcArea(); ✗  
  
calcArea(); ✗  
  
byte h = calcArea(4, 20); ✗  
  
int j = calcArea(2, 3, 5); ✗
```

## Exercise in chapter 4 page 88

A 42 84

B void method has a return statement. it should be String time.

## Exercise in chapter 4 page 89

A class can have any number of these. – instance variables

A method can have only one of these. - return

This can be implicitly promoted. - argument

I prefer my instance variables private. - encapsulation

It really means “make a copy.” – pass by value

Only setters should update these. – instance variables

A method can have many of these. - arguments

I return something by definition. - method

I shouldn't be used with instance variables - public.

I can have many arguments. - method

By definition, I take one argument. - setter

These help create encapsulation. – getter and setter

## Exercise in chapter 4 page 90

**Candidates:**

i < 9  
index < 5

i < 20  
index < 5

i < 7  
index < 7

i < 19  
index < 1

**Possible output:**

14 7

9 5

19 1

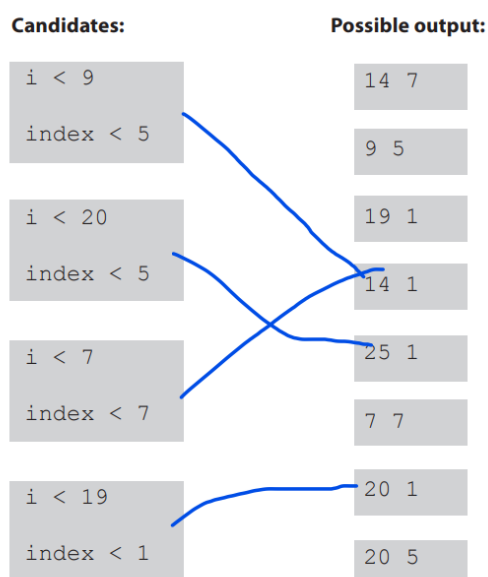
14 1

25 1

7 7

20 1

20 5





## Exercise in chapter 4 page 91

```
public class Puzzle4 {  
    public static void main(String[] args) {  
        Value[] values = new Value[6];  
        int number = 1;  
        int i = 0;  
        while (i < 6) {  
            values[i] = new Value();  
            values[i].intValue = number;  
            number = number * 10;  
            i = i + 1;  
        }  
        int result = 0;  
        i = 6;  
        while (i > 0) {  
            i = i - 1;  
            result = result + values[i].doStuff(i);  
        }  
        System.out.println("result " + result);  
    }  
}  
  
class Value {  
    int intValue;  
    public int doStuff(int factor) {  
        if (intValue > 100) {  
            return intValue * factor;  
        } else {  
            return intValue * (5 - factor);  
        }  
    }  
}
```

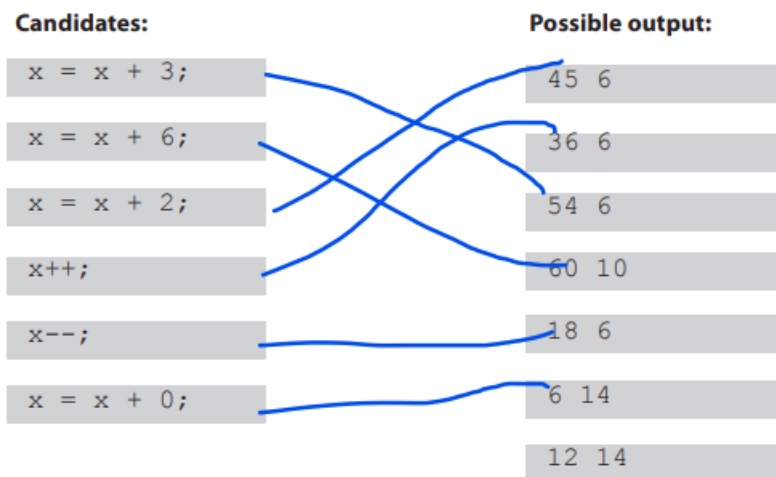
### Exercise in chapter 5 page 118

```
class Output {  
    public static void main(String[] args) {  
        Output output = new Output();  
        output.go();  
    }  
    void go() {  
        int value = 7;  
        for (int i = 1; i < 8; i++) {  
            value++;  
            if (i > 4) {  
                System.out.print(++value + " ");  
            }  
            if (value > 14) {  
                System.out.println(" i = " + i);  
                break;  
            }  
        }  
    }  
}
```

### Exercise in chapter 5 page 119

```
class MultiFor {  
    public static void main(String[] args) {  
        for (int i = 0; i < 4; i++) {  
            for (int j = 4; j > 2; j--) {  
                System.out.println(i + " " + j);  
            }  
            if (i == 1) {  
                i++;  
            }  
        }  
    }  
}
```

## Exercise in chapter 5 page 121



## Exercise in chapter 5 page 163

```
import java.util.ArrayList;

public class ArrayListMagnet {

    public static void main(String[] args) {
        ArrayList<String> a = new ArrayList<String>();
        a.add(0, "zero");
        a.add(1, "one");
        a.add(2, "two");
        a.add(3, "three");
        printList(a);
        if (a.contains("three")) {
            a.add("four");
        }
        a.remove(2);
        printList(a);
        if (a.indexOf("four") != 4) {
            a.add(4, "4.2");
        }
    }
}
```

```

}
printList(a);
if (a.contains("two")) {
a.add("2.2");
}
printList(a);
}
public static void printList(ArrayList<String> list) {
for (String element : list) {
System.out.print(element + " ");
}
System.out.println();
}
}

```


## Exercise in chapter 5 page 164

### Across

- 1.primitive
- 6.object
- 7. indexof
- 9.if
- 12. add
- 13.long
- 15.get
- 17. arraylist
- 19. size
- 21.length
- 22. tapas
- 23. import

### down

- 5.double
- 4.short
- 2. method
- 3. element
- 8.package
- 10. float
- 11. contains
- 15. virtual
- 15. Is empty
- 18 .list
- 20. Api
- 21. loop

 **Sharpen your pencil**

How many instance variables does Surgeon have? 1

How many instance variables does FamilyDoctor have? 2

How many methods does Doctor have? 1

How many methods does Surgeon have? 2


How many methods does FamilyDoctor have? 2

Can a FamilyDoctor do treatPatient()? Y

Can a FamilyDoctor do makeIncision()? N

→ **Yours to solve.**

you are here ▶ 171

 **Sharpen your pencil**

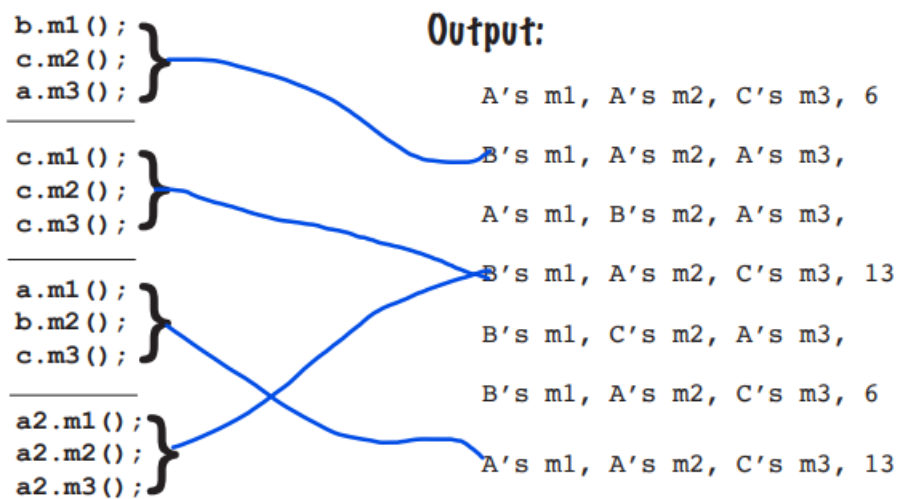
Put a check next to the relationships that make sense.

- ☒ Oven extends Kitchen
- ☒ Guitar extends Instrument
- ☐ Person extends Employee
- ☐ Ferrari extends Engine
- ☒ FriedEgg extends Food
- ☒ Beagle extends Pet
- ☐ Container extends Jar
- ☐ Metal extends Titanium
- ☐ GratefulDead extends Band
- ☒ Blonde extends Smart
- ☐ Beverage extends Martini

*Hint: apply the IS-A test*

you are here ▶ 181

Exercise in page 194



Exercise in page 195

Set 1 works perfectly. Set 2 will not compile due to the wrong return type. Set 3 and 4 will give wrong outputs

### Exercise in page 196

```
public class Rowboat extends Boat {  
    public void rowTheBoat() {  
        System.out.print("stroke natasha");  
    }  
}  
  
public class Boat {  
    private int length ;  
    public void setLength ( int len ) {  
        length = len;  
    }  
    public int getLength() {  
        return length ;  
    }  
    public void move() {  
        System.out.print("drift ");  
    }  
}
```

```

public class TestBoats {
    public static void main(String[] args){
        Boat b1 = new Boat();
        Sailboat b2 = new Sailboat();
        Rowboat b3 = new Rowboat();
        b2.setLength(32);
        b1.move();
        b3.move();
        b2.move();
    }
}

public class Sailboat extends Boat {
    public void move() {
        System.out.print("hoist sail ");
    }
}

```

### Exercise In page 232

```

2 vout - - - > Vin
3 Fluffie → Muffie- - >whuffie
4 Goop → boop→ soup
5   alpha - - > beta
      →Gamma →delta
      - ->epsilon

```

### Exercise in page 233

```

1 public abstract class Top { }
public class Tip extends Top { }

2. public abstract class Fee { }
public abstract class Fi extends Fee { }

3. public interface Foo { }
public class Bar implements Foo { }

```

```

public class Baz extends Bar { }

4 public interface Zeta { }

public class Alpha implements Zeta { }

public interface Beta { }

public class Delta extends Alpha implements Beta { }

```

## exercise in page 236

<pre> interface Nose {     public int iMethod() }  abstract class Picasso implements nose {     public int imethod()         return 7; }  class clown extends picasso { }  class Acts extends Picasso {     public int iMethod(){ </pre>	<pre> public class of76 extends Clowns {      public static void main(String[] args) {         Nose[] i= new Node[3]          i[0] = new acts         i[1] = new picasso         i[2] = new of76          for (int x = 0; x &lt; 3; x++) {             System.out.println( i[x].imethod()                                 + " " + i[x].getClass());         }     } } </pre>
--	--

Exercise in page 237 – honey





## Sharpen your pencil

→ Yours to solve.

Match the new Duck() call with the constructor that runs when that Duck is instantiated. We did the easy one to get you started.

```

public class TestDuck {
    public static void main(String[] args) {
        int weight = 8;
        float density = 2.3F;
        String name = "Donald";
        long[] feathers = {1, 2, 3, 4, 5, 6};
        boolean canFly = true;
        int airspeed = 22;

        Duck[] d = new Duck[7];
        d[0] = new Duck();
        d[1] = new Duck(density, weight);
        d[2] = new Duck(name, feathers);
        d[3] = new Duck(canFly);
        d[4] = new Duck(3.3F, airspeed);
        d[5] = new Duck(false);
        d[6] = new Duck(airspeed, density);
    }
}

```

```

class Duck {
    private int kilos = 6;
    private float floatability = 2.1F;
    private String name = "Generic";
    private long[] feathers = {1, 2, 3, 4, 5, 6, 7};

    private boolean canFly = true;
    private int maxSpeed = 25;

    public Duck() {
        System.out.println("type 1 duck");
    }

    public Duck(boolean fly) {
        canFly = fly;
        System.out.println("type 2 duck");
    }

    public Duck(String n, long[] f) {
        name = n;
        feathers = f;
        System.out.println("type 3 duck");
    }

    public Duck(int w, float f) {
        kilos = w;
        floatability = f;
        System.out.println("type 4 duck");
    }

    public Duck(float density, int max) {
        floatability = density;
        maxSpeed = max;
        System.out.println("type 5 duck");
    }
}

```

Page 254



## Sharpen your pencil

What's the real output? Given the code on the left, what prints out when you run TestHippo? A or B?

(The answer is at the bottom of the page.)

A

```

File Edit Window Help Swear
% java TestHippo
Starting...
Making an Animal
Making a Hippo

```

B

```

File Edit Window Help Swear
% java TestHippo
Starting...
Making a Hippo
Making an Animal

```

## Exercise on page 268

will execute for a long time, giving the Garbage Collector time to do its stuff.)

```
public class GC {
    public static GC doStuff() {
        GC newGC = new GC();
        doStuff2(newGC);
        return newGC;
    }

    public static void main(String[] args) {
        GC gc1;
        GC gc2 = new GC();
        GC gc3 = new GC();
        GC gc4 = gc3;
        gc1 = doStuff();

        // call more methods
    }

    public static void doStuff2(GC copyGC) {
        GC localGC = copyGC;
    }
}
```

**A**

→ Answers on page 272.

- 1 copyGC = null;
- 2 gc2 = null;
- 3 newGC = gc3;
- 4 gc1 = null;
- 5 newGC = null;
- 6 gc4 = null;
- 7 gc3 = gc2;
- 8 gc1 = gc4;
- 9 gc3 = null;

## Page 287

- ① 

```
public class Foo {
    static int x;

    public void go() {
        System.out.println(x);
    }
}
```
- ② 

```
public class Foo2 {
    int x;

    public static void go() {
        System.out.println(x);
    }
}
```
- ③ 

```
public class Foo3 {
    final int x;

    public void go() {
        System.out.println(x);
    }
}
```
- ④ 

```
public class Foo4 {
    static final int x = 12;

    public void go() {
        System.out.println(x);
    }
}
```
- ⑤ 

```
public class Foo5 {
    static final int x = 12;

    public void go(final int x) {
        System.out.println(x);
    }
}
```
- ⑥ 

```
public class Foo6 {
    int x = 12;

    public static void go(final int x) {
        System.out.println(x);
    }
}
```

## Exercise in page 306

### The output is:

super static block

static block 3

in main

super constructor

constructor

## exercise in 308

1. To use the Math class, the first step is to make an instance of it. ✗
2. You can mark a constructor with the **static** keyword. ✗
3. Static methods don't have access to instance variable state of the "this" object. ✓
4. It is good practice to call a static method using a reference variable. ✗
5. Static variables could be used to count the instances of a class. ✓
6. Constructors are called before static variables are initialized. ✗
7. MAX\_SIZE would be a good name for a static final variable. ✓
8. A static initializer block runs before a class's constructor runs. ✓
9. If a class is marked final, all of its methods must be marked final. ✗
10. A final method can be overridden only if its class is extended. ✗
11. There is no wrapper class for boolean primitives. ✗
12. A wrapper is used when you want to treat a primitive like an object. ✓
13. The parseXxx methods always return a String. ✗
14. Formatting classes (which are decoupled from I/O) are in the java.format package. ✗

## Exercise in page 334

Comparable

comapreTo()

yes

yes

no

ni

comparator

compare()

### **exercise on page 365**

```
import java.util.*;
```

```
public class SortMountains {
```

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

```
        new SortMountains().go();
```

```
    }
```

```
    public void go() {
```

```
        List<Mountain> mountains = new ArrayList<>();
```

```
        mountains.add(new Mountain("Longs", 14255));
```

```
        mountains.add(new Mountain("Elbert", 14433));
```

```
        mountains.add(new Mountain("Maroon", 14156));
```

```
        mountains.add(new Mountain("Castle", 14265));
```

```
        System.out.println("as entered:\n" + mountains);
```

```
        mountains.sort((mount1, mount2) ->
```

```
mount1.name.compareTo(mount2.name));
```

```
        System.out.println("by name:\n" + mountains);
```

```
        mountains.sort((mount1, mount2) -> mount2.height - mount1.height);
```

```
        System.out.println("by height:\n" + mountains);
```

```
    }
```

```
}
```

```
class Mountain {
```

```
    String name;
```

```
    int height;
```

```
    Mountain(String name, int height) {
```

```
        this.name = name;
```

```

        this.height = height;
    }

    public String toString() {
        return name + " " + height;
    }
}

```

### Exercise in page 342

```

songList.sort((one, two) -> one.getBpm() - two.getBpm());
songList.sort((one, two) -> two.getTitle().compareTo(one.getTitle()));

```

### exercise in page 363

- ☒ `takeAnimals(new ArrayList<Animal>());`
- ☐ `takeDogs(new ArrayList<Animal>());`
- ☐ `takeAnimals(new ArrayList<Dog>());`
- ☒ `takeDogs(new ArrayList<>());`
- ☒ `List<Dog> dogs = new ArrayList<>();`  
`takeDogs(dogs);`
- ☒ `takeSomeAnimals(new ArrayList<Dog>());`
- ☒ `takeSomeAnimals(new ArrayList<>());`
- ☒ `takeSomeAnimals(new ArrayList<Animal>());`
- ☒ `List<Animal> animals = new ArrayList<>();`  
`takeSomeAnimals(animals);`
- ☐ `List<Object> objects = new ArrayList<>();`  
`takeObjects(objects);`
- ☐ `takeObjects(new ArrayList<Dog>());`
- ☒ `takeObjects(new ArrayList<Object>());`

## Exercise in page 372

### Candidates:

```
for (int i = 1; i < nums.size(); i++)
    output += nums.get(i) + " ";
```

```
for (Integer num : nums)
    output += num + " ";
```

```
for (int i = 0; i <= nums.length; i++)
    output += nums.get(i) + " ";
```

```
for (int i = 0; i <= nums.size(); i++)
    output += nums.get(i) + " ";
```

### Possible output:

```
1 2 3 4 5
```

```
Compiler error
```

```
2 3 4 5
```

```
Exception thrown
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

```
[1, 2, 3, 4, 5]
```

## Exercise in page 374

filter

skip

limit

distinct

sorted

map

dropWhile

takeWhile

Changes the current element in the stream into something else

Sets the maximum number of elements that can be output from this Stream

While a given criteria is true, will not process elements

Only allows elements that match the given criteria to remain in the Stream

Will only process elements while the given criteria is true

States the result of the stream should be ordered in some way

This is the number of elements at the start of the Stream that will not be processed

Use this to make sure duplicates are removed

### Exercise ein page 395

```
☒ Runnable r = () -> System.out.println("Hi!");  
☒ Consumer<String> c = s -> System.out.println(s);  
☐ Supplier<String> s = () -> System.out.println("Some string");  
☐ Consumer<String> c = (s1, s2) -> System.out.println(s1 + s2);  
☐ Runnable r = (String str) -> System.out.println(str);  
☒ Function<String, Integer> f = s -> s.length();  
☒ Supplier<String> s = () -> "Some string";  
☐ Consumer<String> c = s -> "String" + s;  
☐ Function<String, Integer> f = (int i) -> "i = " + i;  
☒ Supplier<String> s = s -> "Some string: " + s;  
☐ Function<String, Integer> f = (String s) -> s.length();
```

### Exercise in page 416

```
public class StreamPuzzle {  
    public static void main(String[] args) {  
        SongSearch songSearch = new SongSearch();  
        songSearch.printTopFiveSongs();  
        songSearch.search("The Beatles");  
        songSearch.search("The Beach Boys");  
    }  
}  
  
class SongSearch {  
    private final List<Song> songs =  
        new JukeboxData.Songs().getSongs();  
    void printTopFiveSongs() {  
        List<String> topFive = songs.stream()  
            .sorted(Comparator.comparingInt(Song::getTimesPlayed))
```

```

.map(song -> song.getTitle())
.limit(5)
.collect(Collectors.toList());
System.out.println(topFive);
}

void search(String artist) {
Optional<Song> result = songs.stream()
.filter(song -> song.getArtist().equals(artist))
.findFirst();
if (result.isPresent()) {
System.out.println(result.get().getTitle());
} else {
System.out.println("No songs found by: " + artist);
}
}

```

Exercise in page 455

```

class MyEx extends Exception { }

public class ExTestDrive {

    public static void main(String[] args) {
        String test = args[0];
        try {
            System.out.print("t");
            doRisky(test);
            System.out.print("o");
        } catch (MyEx e) {
            System.out.print("a");
        } finally {
            System.out.print("w");
        }
        System.out.println("s");
    }
}

```



```

}

static void doRisky(String t) throws MyEx {

System.out.print("h");

if ("yes".equals(t)) {

throw new MyEx();

}

System.out.print("r");

}

}

```

### Exercise in page in 454

1. A try block must be followed by a catch and a finally block. ✗
2. If you write a method that might cause a compiler-checked exception, you must wrap that risky code in a try/catch block. ✗, can
3. Catch blocks can be polymorphic. ✓
4. Only "compiler checked" exceptions can be caught. ✗, runtime
5. If you define a try/catch block, a matching finally block is optional. ✓
6. If you define a try block, you can pair it with a matching catch or finally block, or both. ✗
7. If you write a method that ~~declares~~ that it can throw a compiler-checked exception, you must also wrap the exception throwing code in a try/catch block. ✗
8. The main() method in your program must handle all unhandled exceptions thrown to it. ✗
9. A single try block can have many different catch blocks. ✓
10. A method can throw only one kind of exception. ✗ many
11. A finally block will run regardless of whether an exception is thrown. ✓
12. A finally block can exist without a try block. ✗ can't
13. A try block can't exist by itself, without a catch block or a finally block. ✗
14. Handling an exception is sometimes referred to as "ducking". ✗
15. The order of catch blocks never matters. ✓
16. A method with a try block and a finally block can ~~optionally~~ declare a checked exception. ✗
17. Runtime exceptions must be handled or declared. ✗

### **Exercise in page 495**

2. +3,0

3. -3,0

4 -3,-3

5 3,3

6. -3,3

### **Exercise in page 504**

1.Jframe

2 listener

3 actionPerformed()

4 setsize()

5 paint component

6 event

7. swing components

8. event object

9 event source

10 addXxxListener( )

11 paintComponent( )

12 Graphics2D

13 repaint( )

14 javax.swing

### **Exercise on page 505**

No class was sent in - button.addActionListener(new ButtonListener());

Actionlistener is a interface therefore implements keyword must be used

### Exercise in page 506

```
import javax.swing.*;
import java.awt.*;
import java.util.concurrent.TimeUnit;

public class Animate {
    int x = 1;
    int y = 1;

    public static void main(String[] args) {
        Animate gui = new Animate ();
        gui.go();
    }

    public void go() {
        JFrame frame = new JFrame();
        frame.setDefaultCloseOperation(
            JFrame.EXIT_ON_CLOSE);
        MyDrawP drawP = new MyDrawP();
        frame.getContentPane().add(drawP);
        frame.setSize(500, 270);
        frame.setVisible(true);
        for (int i = 0; i < 124; i++,y++,x++ ) {
            x++;
            drawP.repaint();
            try {
                TimeUnit.MILLISECONDS.sleep(50);
            } catch (Exception ex) { }
        }
    }

    class MyDrawP extends JPanel {
        public void paintComponent(Graphics g ) {
            g.setColor(Color.white);
```

```

g.fillRect(0, 0, 500, 250);

g.setColor(Color.blue);

g.fillRect(x, y, 500-x*2, 250-y*2);

```

## exercise in page 535

**A**

```

JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(button);
frame.getContentPane().add(BorderLayout.NORTH, buttonTwo);
frame.getContentPane().add(BorderLayout.EAST, panel);

```

4

---

**B**

```

JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);
frame.getContentPane().add(BorderLayout.EAST, panel);

```

6

---

**C**

```

JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);

```

1

---

**D**

```

JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.NORTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.CENTER, button);

```

2

---

**E**

```

JFrame frame = new JFrame();
JPanel panel = new JPanel();
panel.setBackground(Color.darkGray);
JButton button = new JButton("tesuji");
JButton buttonTwo = new JButton("watari");
frame.getContentPane().add(BorderLayout.SOUTH, panel);
panel.add(buttonTwo);
frame.getContentPane().add(BorderLayout.NORTH, button);

```

3

## EXERCISE IN PAGE 582

1. Serialization is appropriate when saving data for non-Java programs to use. **F**
2. Object state can be saved only by using serialization. **F**
3. ObjectOutputStream is a class used to save serializable objects. **T**
4. Chain streams can be used on their own or with connection streams. **F**
5. A single call to writeObject() can cause many objects to be saved. **T**
6. All classes are serializable by default. **F**
7. The java.nio.file.Path class can be used to locate files. **F**
8. If a superclass is not serializable, then the subclass can't be serializable. **F**
9. Only classes that implement AutoCloseable can be used in try-with-resources statements. **T**
10. When an object is deserialized, its constructor does not run. **T**
11. Both serialization and saving to a text file can throw exceptions. **T**
12. BufferedWriters can be chained to FileWriters. **T**
13. File objects represent files, but not directories. **F**
14. You can't force a buffer to send its data before it's full. **F**
15. Both file readers and file writers can optionally be buffered. **T**
16. The methods on the Files class let you operate on files and directories. **T**
17. Try-with-resources statements cannot include explicit finally blocks. **F**

## EXERCISE IN PAGE 583

```
import java.io.*;

class DungeonGame implements Serializable {
    public int x = 3;
    transient long y = 4;
    private short z = 5;
    int getX() {
        return x;
    }
}
```

```


long getY() {
    return y;
}

short getZ() {
    return z;
}
}

class DungeonTest {
    public static void main(String[] args) {
        DungeonGame d = new DungeonGame();
        System.out.println(d.getX() + d.getY() + d.getZ());
        try {
            FileOutputStream fos = new FileOutputStream("dg.ser");
            ObjectOutputStream oos = new ObjectOutputStream(fos);
            oos.writeObject(d);
            oos.close();

            FileInputStream fis = new FileInputStream("dg.ser");
            ObjectInputStream ois = new ObjectInputStream(fis);
            d = (DungeonGame) ois.readObject();
            ois.close();
        } catch (Exception e) {
            e.printStackTrace();
        }
        System.out.println(d.getX() + d.getY() + d.getZ())
    }
}

```

 **Sharpen your pencil** —————

**Fill in the blanks:** ————— **→ Yours to solve.**

What two pieces of information does the client need in order to make a connection with a server? IP address TCP port number

Which TCP port numbers are reserved for "well-known services" like HTTP and FTP? 0 to 1023

TRUE or FALSE: The range of valid TCP port numbers can be represented by a short primitive. yes

### Exercise in page 631

1. ExecutorService
2. SocketChannel, Socket
3. InterruptedException,
4. Thread pool
5. IP Address, Host name, port
6. Thread
7. Executors
8. ServerSocketChannel
9. Thread.sleep(), CountdownLatch
10. Runnable
11. InetSocketAddress

### Exercise in page 668

Synchronized instead of static so that no data are lost as only one thread runs at a time