Object-Oriented Programming In Mechatronic Systems

Summer School 2018

Module 4 – Advanced Programming in Java
Aachen, Germany

Cybernetics Lab IMA & IfU
Faculty of Mechanical Engineering
RWTH Aachen University

Recap

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 | Aachen, Germany | Cybernetics Lab IMA & IfU

Recap

```java
public class IfElseDemo {
    public static void main(String[] args) {
        int testscore = 76;
        char grade;

        if (testscore >= 90) {
            grade = 'A';
        } else if (testscore >= 80) {
            grade = 'B';
        } else if (testscore >= 70) {
            grade = 'C';
        } else {
            grade = 'F';
        }
        System.out.println("Grade = " + grade);
    }
}
```

```
public class LoopDemo{

    public static void main(String[] args) {

        for (int i = 0; i < 15; i++){

            System.out.println("Loop " + i);

        }

}
```

i = i + 1

Output:
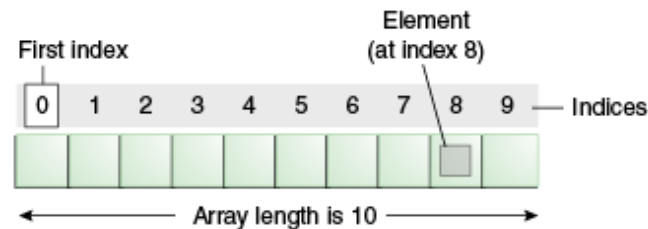Loop 0
Loop 1
…
Loop 14

IfU IMA RWTH AACHEN UNIVERSITY

Recap and Motivation

- Primitive data types (e.g. `int`) can only hold a single value
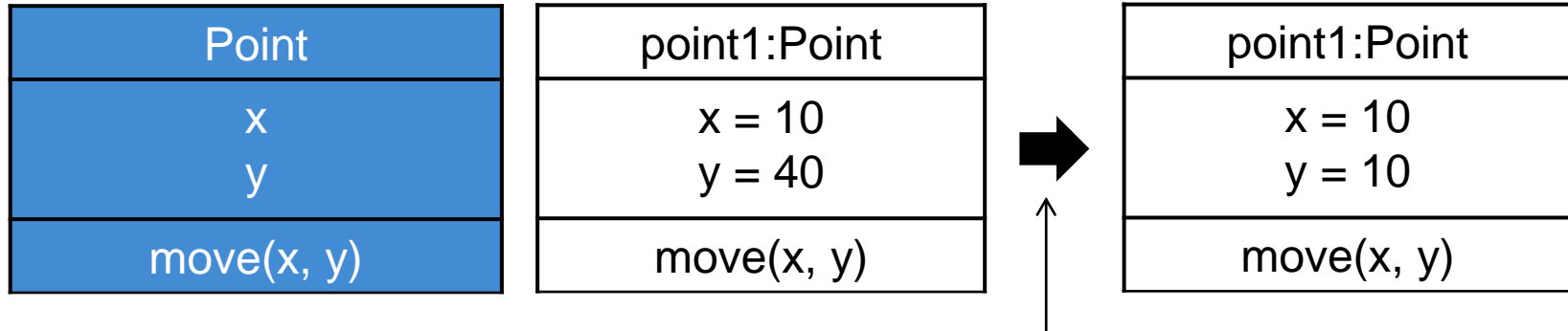- E.g. `int val = 17;`

Array Features

- Arrays can hold multiple values (or *elements*)!
- Can only hold one data type, i.e. no mixture of data types (e.g. `int` **and** `char`)
- Length is established upon creation: `int[] numbers = new int[10]`
- After that it's fixed!
- Access to elements via *index:* `numbers[0], numbers[5]`
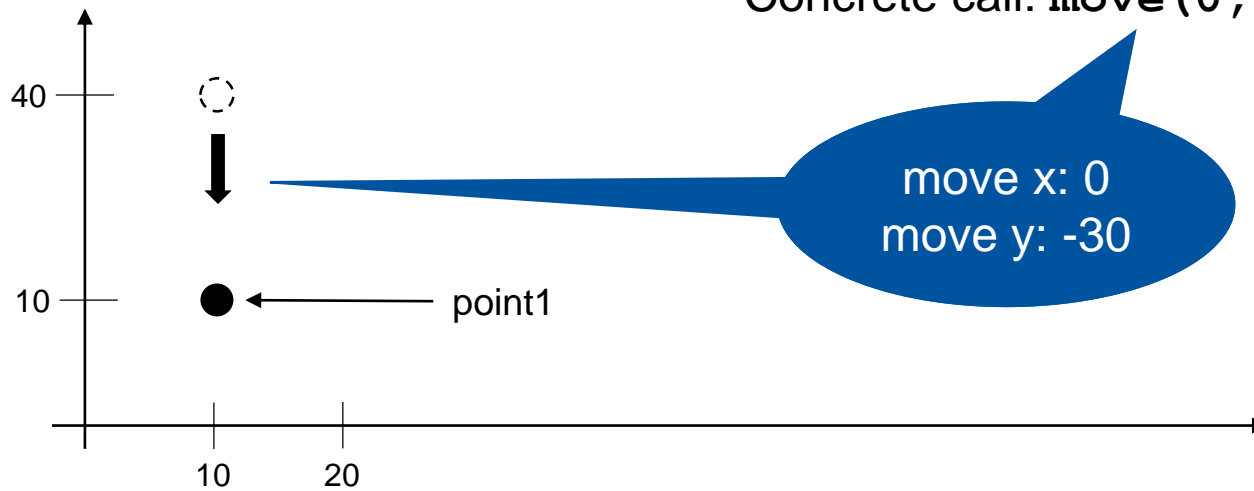- Index starts with 0. That is, the first array element has the index 0:

Example: Classes and Objects

Point
x   y
move(x, y)

point1:Point
x = 10   y = 40
move(x, y)

point1:Point
x = 10   y = 10
move(x, y)

Concrete call: `move(0, -30)`

move x: 0
move y: -30

40
10
point1
10   20

IfU   IMA   RWTH AACHEN UNIVERSITY

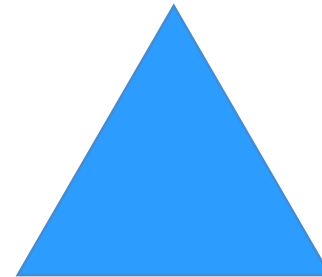## Instances: Example

```
public class Shape {

    public String color;

    public boolean filled;

}
```


This **Rectangle** is a **Shape**
with color "red" and not filled

This **Triangle** is a **Shape**
with color "blue" and filled

Constructors

- Constructors run *before* the object can be assigned to a reference
- It runs every time you invoke `new`
- If you don't write a constructor for your class the compiler writes one for you …
- … which is called the **default constructor**!

Example

```
public class Circle {
    private int radius;
    public Circle(){

    }

    public Circle(int r){
        this.radius = r;

    }
}
```

Default constructor

Constructor

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 |
Aachen, Germany | Cybernetics Lab IMA & IfU

Communication and Relations between Objects

Communication and Relations between Objects

Point2D
-x:double -y:double
+Point2D(x:double, y:double) +getX():double +getY():double +move(delta_x:double, delta_y:double)

Every **rectangle** is defined by two **Point2Ds**:
- **First point** defines the lower left corner
- **Second point** defines the upper right corner

Communication and Relations between Objects

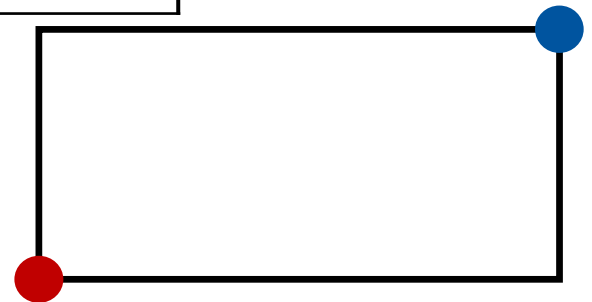| Rectangle |
|---|
| **-lowerLeft:Point2D**
-upperRight:Point2D |
| +Rectangle(**lowerLeft:Point2D**, **upperRight:Point2D**)

+setLowerLeft(lowerLeft:Point2D):void
+getLowerLeft():Point2D

+setUpperRight(upperRight:Point2D):void
+getUpperRight():Point2D |

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 | Aachen, Germany | Cybernetics Lab IMA & IfU

Communication and Relations between Objects

Live Demo

```java
public class Rectangle {

    private Point2D lowerLeft;
    private Point2D upperRight;

    public Rectangle(Point2D lowerLeft, Point2D upperRight) {
        this.lowerLeft = lowerLeft;
        this.upperRight = upperRight;
    }
    …
}
```

Constructor

lowerLeft and **upperRight** are names of **local and instance variables**!
Hence, we need a way to distinguish the variables ➔ `this`

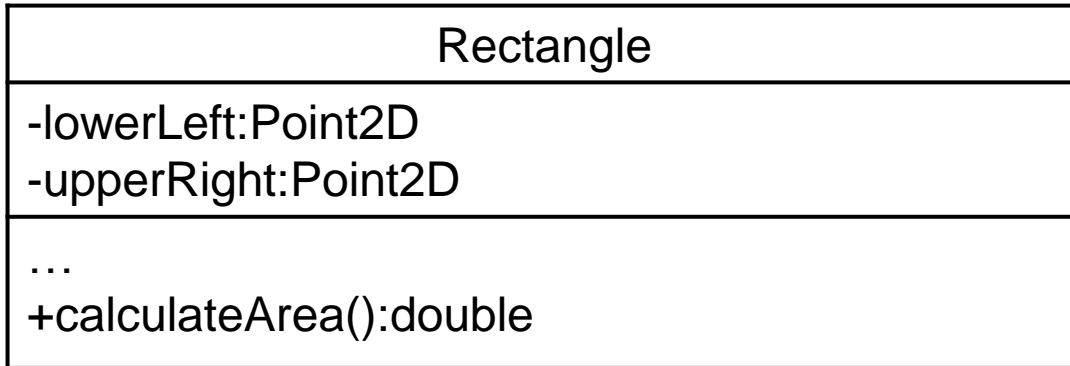# Communication and Relations between Objects

`this` represents a reference (a pointer) to the current object

```java
public class Rectangle {
    private Point2D lowerLeft;

    private Point2D upperRight;
    …
    public void setLowerLeft(Point2D lowerLeft) {
        this.lowerLeft = lowerLeft;
    }

    public Point2D getLowerLeft() {
        return lowerLeft;
    }
    …
}
```

Communication and Relations between Objects

Rectangle
-lowerLeft:Point2D -upperRight:Point2D
… +calculateArea():double
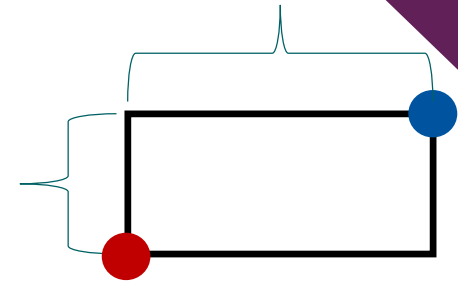
```
public class Rectangle {
    …
    public double calculateArea() {
        return (upperRight.getX() - lowerLeft.getX()) *
            (upperRight.getY() - lowerLeft.getY());
    }
}
```

How can we ensure that this calculation works?

More precisely: How can we ensure that lower left and upper right represents these points and not others?

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 |
Aachen, Germany | Cybernetics Lab IMA & IfU

Communication and Relations between Objects

Live Demo

We have to check, if the following conditions are always satisfied:

- lowerLeft.getX() < upperRight.getX() **and**
- lowerLeft.getY() < upperRight.getY()

```
public class Rectangle {
    private Point2D lowerLeft;
    private Point2D upperRight;
    …
    public void setLowerLeft(Point2D lowerLeft) {
        if (lowerLeft.getX() < upperRight.getX() &&
            lowerLeft.getY() < upperRight.getY()) {

            this.lowerLeft = lowerLeft;
        }
        // else we do nothing (perhaps not the best solution!)
    }
}
```
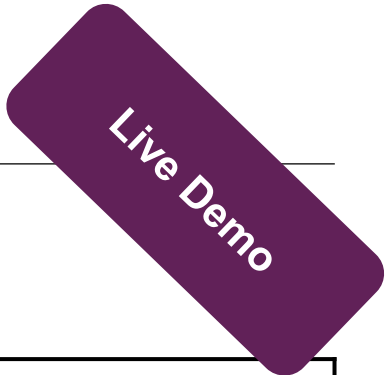
Live Demo

```java
public class Application {

    public static void main(String[] args) {
        Rectangle rect = new Rectangle(
            new Point2D(10,10),
            new Point2D(20,40)
        );
        System.out.println("The area of rect is: " +
            rect.calculateArea();
    }
}
```

Communication and Relations between Objects

Rectangle
-**lowerLeft:Point2D** -**upperRight:Point2D**
+Rectangle(**lowerLeft:Point2D**, **upperRight:Point2D**) +**getLowerLeft():Point2D** +**getUpperRight():Point2D** +**setLowerLeft(lowerLeft:Point2D):void** +**setUpperRight(upperRight:Point2D):void**

Square
+Square(**lowerLeft:Point2D**, **upperRight:Point2D**)

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 |
Aachen, Germany | Cybernetics Lab IMA & IfU
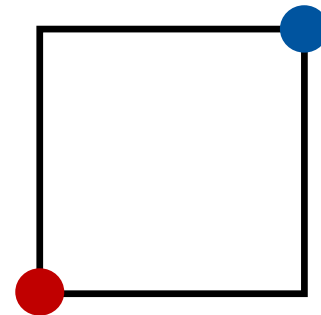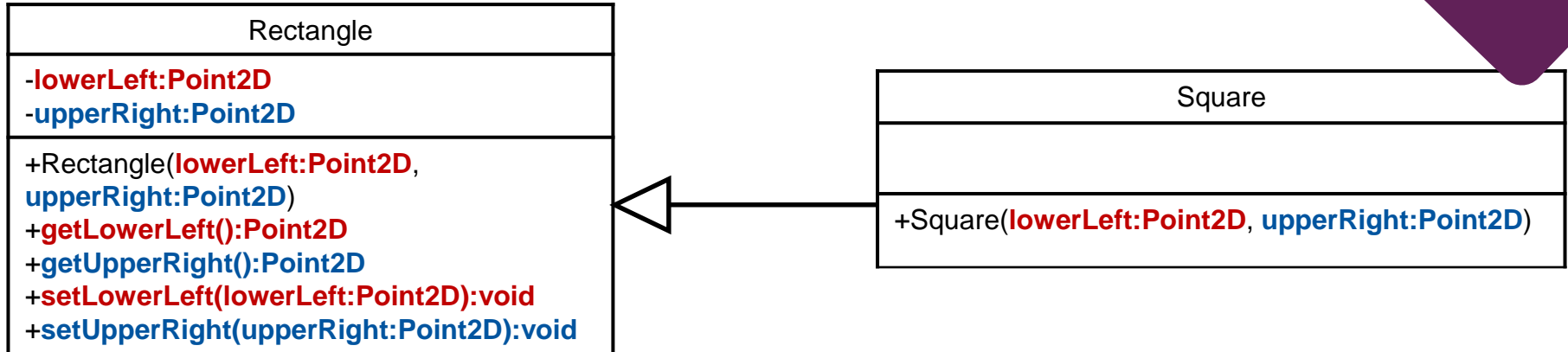
Communication and Relations between Objects

```
public class Square extends Rectangle {
        …
    public Square(Point2D lowerLeft, Point2D upperRight){
        super(lowerLeft, upperRight);

        if(this.upperRight.getX() – this.lowerLeft.getX()
            != this.upperRight.getY() – this.lowerLeft.getY())
        {
            throw new IllegalStateException();
        }
    }
}
```
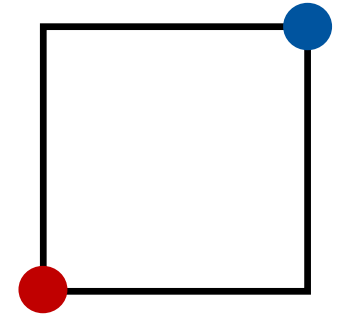
A Square is a special case of a rectangle, where the sides have equal length
- Square extends and specializes Rectangle
- To save the two points needed to describe a rectangle, we need to call the constructor of Rectangle
- Use **super** to call the overridden method or constructor of a **superclass**

Object-Oriented Programming In Mechatronic Systems | Summer School 2018 | Aachen, Germany | Cybernetics Lab IMA & IfU

Exception Handling

How to handle unforeseen events during program execution?

- That is **during runtime**: Events like server down, file not found…
- Not everything is under your control! E.g., can you control an external server?
- Such events are called *Exception*
- Shorthand for exceptional event

```
public class Application {

    public static void main(String[] args) {
        Rectangle r = new Rectangle(new Point2D(50,10), new
            Point2D(20,40));
        System.out.println("The area of r is: " +
            r.calculateArea();
    }
}
```

This would result in an illegal state!

```
lowerLeft.getX() < upperRight.getX() &&
lowerLeft.getY() < upperRight.getY()
```
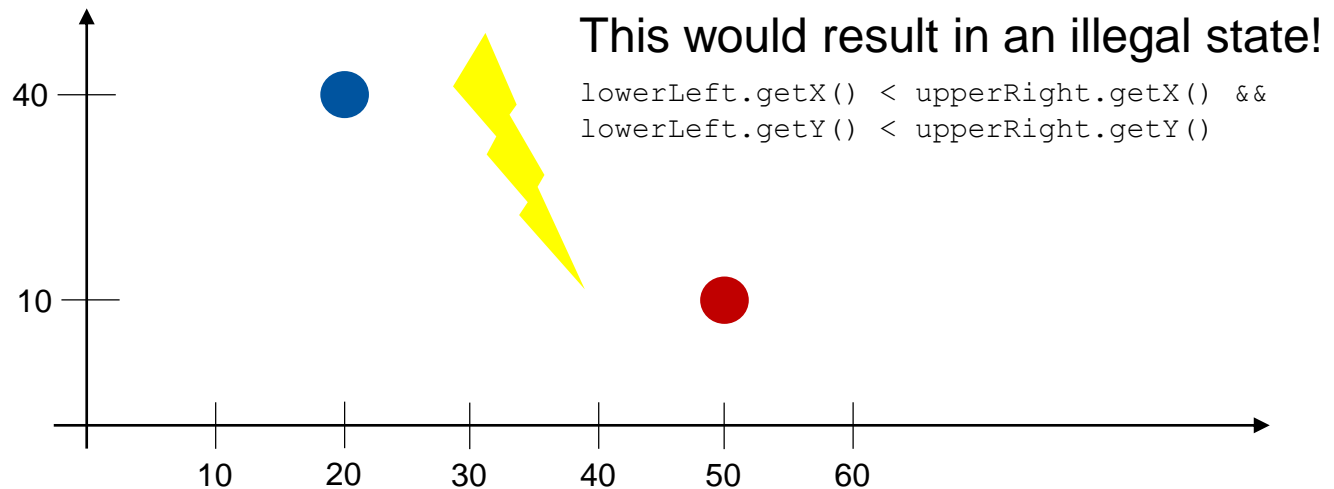
```
public class Application {

    public static void main(String[] args) {
        try {
            Rectangle r = new Rectangle(new Point2D(50,10), new
                Point2D(20,40));
            System.out.println("The area of r is: " +
                r.calculateArea();
        } catch (IllegalStateException e) {
            System.err.println("The initialization of rectangle
                failed. Reason: " + e.getMessage();
        }
    }
}
```

If the initialization fails (due to a created illegal state), an
`IllegalStateException` is thrown: Now, we can react accordingly, by catching
the `Exception`.

Exception Handling

How are exceptions handled in Java?

- If exception occurs within a method, the method throws an *exception object*
- This object contains information about the error such as type and message
- The runtime tries to find something to handle it, by searching the method call stack for an *exception handler*!
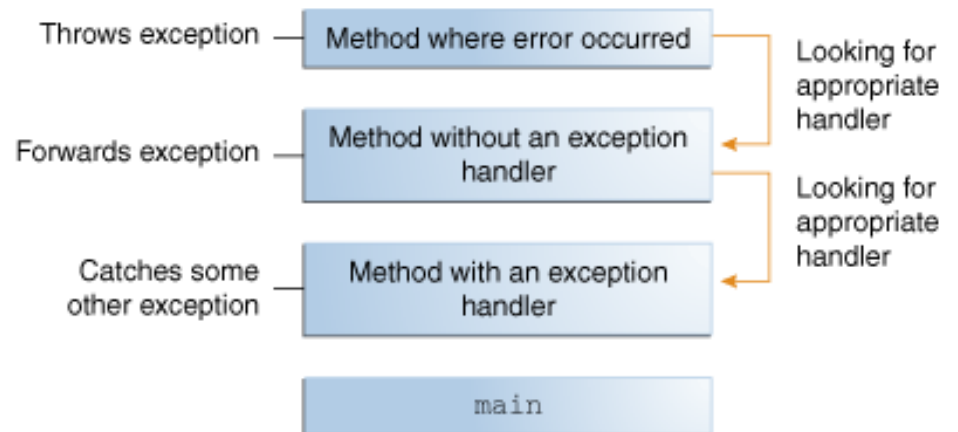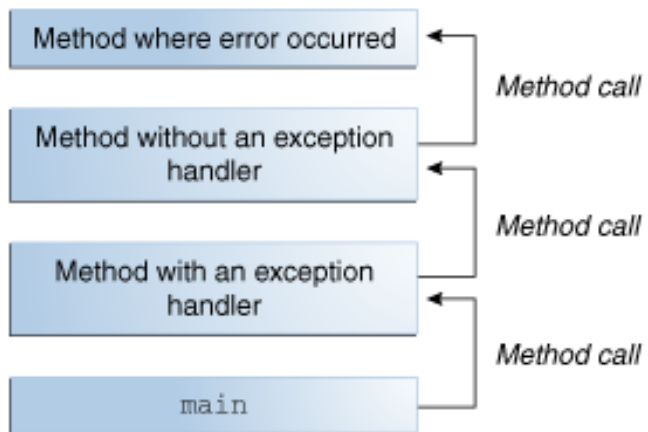
The Catch or Specify Requirement

- Valid code must honor the *Catch or Specify* Requirement

- If code might throw certain exceptions, code **must** be enclosed by…

  ➢ … a `try` statement that catches the exception **or**

  ➢ … a method that is marked via the `throws` clause (telling the caller that the method can throw such exceptions)

- Code that does not honor the requirement doesn't compile!

Catching and Handling Exceptions

Three exception handler components: `try`, `catch`, and `finally`

```
try
{
    statements that can throw exceptions
}
catch (exception-type identifier)
{
    statements executed when exception is thrown
}
finally // not mandatory!
{
    statements that are always executed
}
```

Exception Handling

```java
public class Rectangle {
    private Point2D lowerLeft;
    private Point2D upperRight;

    …

    public void setUpperRight(Point2D upperRight)
        throws IllegalStateException {
        if (lowerLeft == null ||
             (lowerLeft.getX() < upperRight.getX() &&
              lowerLeft.getY() < upperRight.getY())
          ) {
            this.upperRight = upperRight;
        } else {
            throw new IllegalStateException("Upper right condition is
                not guaranteed");
        }
    }

    …

}
```

Live Demo

```java
public class Rectangle {
    private Point2D lowerLeft;
    private Point2D upperRight;

    …

    public void setUpperRight(Point2D upperRight)
        throws IllegalStateException, IllegalArgumentException {
        if (upperRight == null) {
            throw new IllegalArgumentException("The argument cannot be
                null");
        }
        if (lowerLeft == null ||
                (lowerLeft.getX() < upperRight.getX() &&
                 lowerLeft.getY() < upperRight.getY())
            ) {
            this.upperRight = upperRight;
        } else {
            throw new IllegalStateException("Upper right condition is
                not guaranteed");
        }
    }

    …

}
```

IfU   IMA   RWTH AACHEN UNIVERSITY

Live Demo

Now we can update our constructor using our setters (with the extended validation check)

```java
public class Rectangle {

    private Point2D lowerLeft;

    private Point2D upperRight;

    // constructor
    public Rectangle(Point2D lowerLeft, Point2D upperRight)
        throws IllegalStateException, IllegalArgumentException {

        setLowerLeft(lowerLeft);
        setUpperRight(upperRight);

    }
    ...
}
```

Live Demo

Finally, we get our application

```java
public class Application {

    public static void main(String[] args) {
        try {
            Rectangle r = new Rectangle(new Point2D(50,10), new
                Point2D(20,40));
            System.out.println("The area of r is: " +
                r.calculateArea());
        } catch (IllegalStateException e1) {
            System.err.println("The initialization of rectangle
                failed. Reason: " + e1.getMessage());
        } catch (IllegalArgumentException e2) {
            System.err.println(e2.getMessage());
        }
    }
}
```

The Java API

Java comes with hundreds of classes

- They are bundled in the Java API
- API is shorthand for *Application Programming Interface*
- What's in it and how do we use the library?
- **In it**: Libs for mathematical operations or data structures like array list …
- **Use it**: You have to know which package the class is in!

Java Packages

- A *package* is a grouping of related types (e.g. classes or interfaces)
- Make stuff easier to find and use
- … to avoid naming conflicts
- … to control access.
- E.g. `java.util` (for utilities) or `javax.swing` (for creating GUIs)

Live Demo

Hundreds of packages and classes (Excerpt)

Thorough description: http://docs.oracle.com/javase/8/docs/api/

Inspect: System.out.println(String s)

Data Structures

Data Structures

What are data structures?

- Informal: a container that provides storage for data items …
- and capabilities for storing and retrieving them.
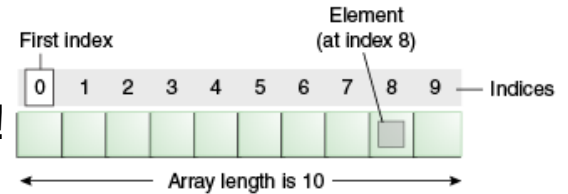- Examples: arrays, linked lists, trees, queues …
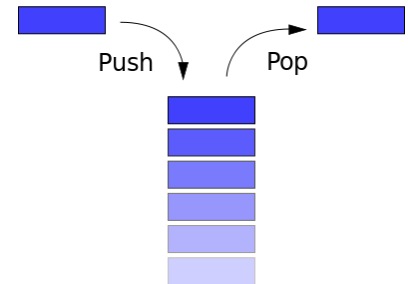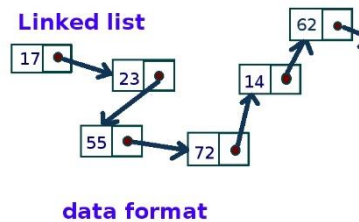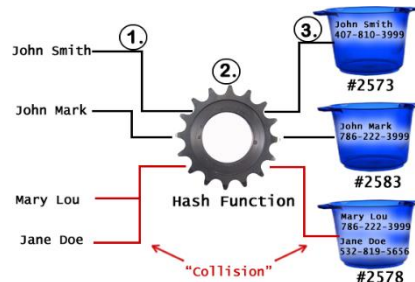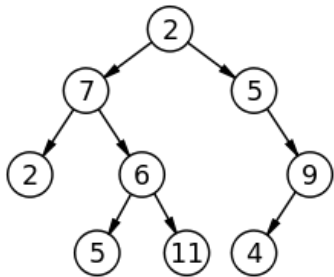
Data structures in Java

- **There are plenty of data structures in the Java API**
- We'll have a look on **array lists and maps**.

Recap: Arrays

- Arrays can hold multiple values (or *elements*)!
- Length is established upon creation, after that it's **fixed**!
- Access to elements via *index,* which starts with 0

## Example

```java
public static void main(String[] args) {
        int[] ar = new int[3];
        ar[0] = 100;
        ar[1] = 200;
        ar[2] = 300;
        System.out.println("Array value on pos 1:" +ar[0]);
        System.out.println("Array value on pos 2:" +ar[1]);
        System.out.println("Array value on pos 3:" +ar[2]);
}
```
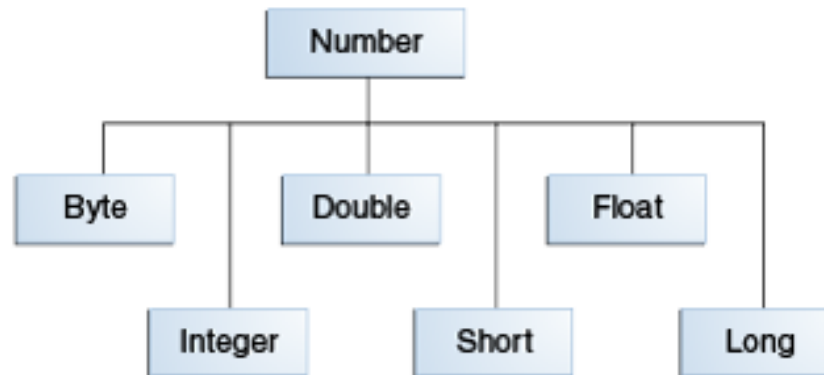
ArrayLists

- Class `java.util.ArrayList`
- `ArrayList` class extends `AbstractList` and implements the `List` interface
- Are created with an initial size
- **Can hold objects** (e.g. class `Integer` or class `Point`)!
- **When this size is exceeded, the collection is automatically enlarged.**
- When objects are removed, the array may be shrunk.
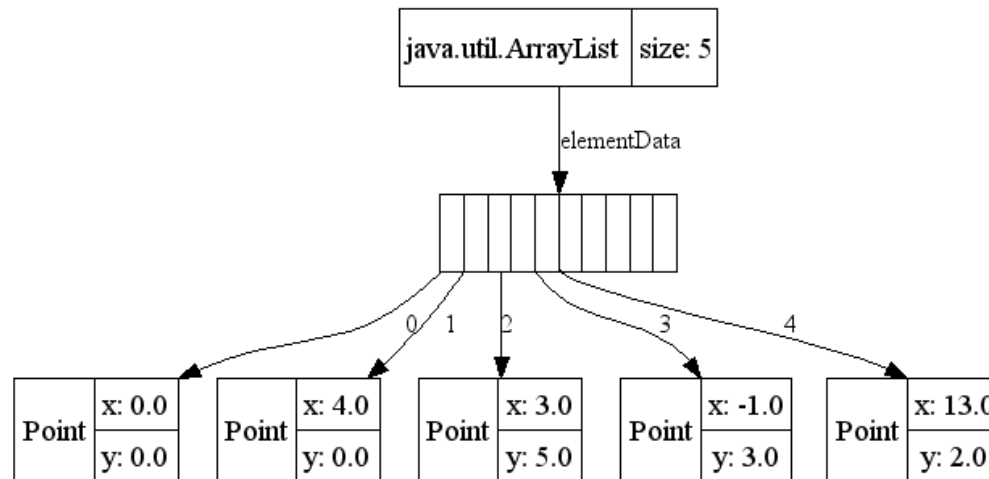
Contrast to arrays!

Excursus: Wrapper Classes

- When working with numbers one can work with primitive types (e.g. `int`).
- There can be reasons to use objects in place of primitives
- Java provides *wrapper classes* for each of the primitive data types.
- Wrapping can be done by compiler (compiler *boxes* primitive in its wrapper class)
- Vice versa: if number object when a primitive is expected (compiler *unboxes*)
- Each wrapper class comes with methods, e.g. `Integer.parseInt("9")`

Data Structures

`ArrayList` Methods (Excerpt)

- **`void add(Object element)`**
  Inserts the specified element at the end of this list.

- **`void clear()`**
  Removes all of the elements from the ArrayList.

- **`int indexOf(Object o)`**
  Returns the index in this list of the first occurrence of the specified element, or -1
  if the List does not contain this element.

- **`Object remove(int index)`**
  Removes the element at the specified position in this list.

- **`int size()`**
  Returns the number of elements in this list.

Live Demo

Example

```java
import java.util.ArrayList;

public class Application {

    public static void main(String[] args) {

        // create new ArrayList to store rectangles
        ArrayList<Rectangle> rList = new ArrayList<>();

        // add three rectangle to list
        rList.add(new Rectangle(new Point2D(0,0), new Point2D(10,10)));
        rList.add(new Rectangle(new Point2D(5,3), new Point2D(6,7)));
        rList.add(new Rectangle(new Point2D(12,13),
            new Point2D(15,18)));

        // continue next slide
```

IfU   IMA   RWTH AACHEN UNIVERSITY

Live Demo

Example (cont'd)

```java
    // get size and display.
    int count = rList.size();
    System.out.println("Number of rectangles: " + count);

    // loop through elements.
    for (int i = 0; i < rList.size(); i++) {
        double x1 = rList.get(i).getLowerLeft().getX();
        double x2 = rList.get(i).getUpperRight().getX();
        System.out.println("x-Range of rectangle " + i + " is "
            + (x2 - x1));
    }
  }
}
```

IfU IMA RWTH AACHEN UNIVERSITY

Thank you very much!