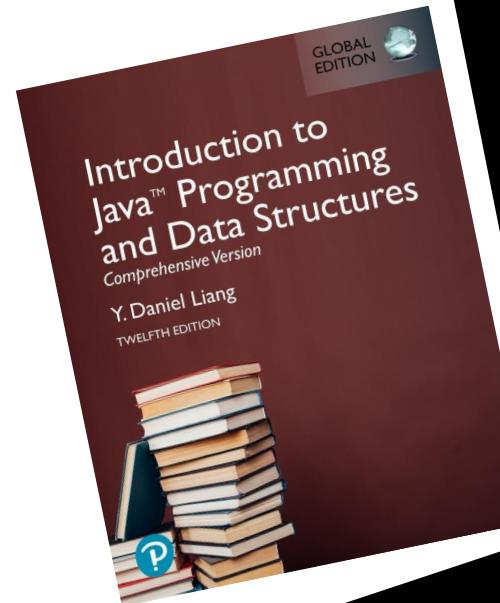
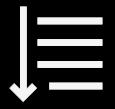


Collections and Maps

Programming 2



Agenda



1. Wrappers
2. Collection
3. List
4. Collections and Sorting
5. Set
6. Map

Example code:

<https://gitlab.com/kdg-ti/programming-1.2/java/examples/m1-collections>

Wrappers

Wrapper classes

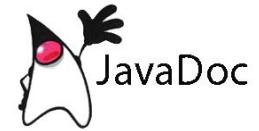


- Every primitive type has a corresponding class

- methods
- constants
- objects

Primitive type	Wrapper class
boolean	Boolean
byte	Byte
char	Character
float	Float
int	Integer
long	Long
short	Short
double	Double

Wrapper method examples

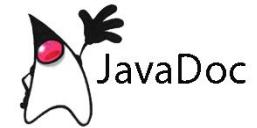


Method	Meaning
Integer Integer.valueOf(String s)	each wrapper has valueOf methods for converting String, primitive... to wrapper
double Double.parseDouble(String s)	parseXXX methods behave like valueOf
String Double.toHexString(double d)	convert to hexadecimal string

Remark:

- valueOf-methods return a wrapper type reference variable (object)
example: `Integer iObj = Integer.valueOf("456");`
- parse-methods return a primitive type variable
example: `int i = Integer.parseInt("123");`

Wrapper constant examples



Constant	Meaning
<code>Integer.MAX_VALUE</code>	214783647. All wrappers have MAX_VALUE and MIN_VALUE
<code>Double.NaN</code>	Not A Number
<code>Double.NEGATIVE_INFINITY</code>	- ∞

Wrappers: boxing/unboxing



- You can explicitly convert between primitives and wrappers, using casting

```
private static void casting() {  
    int nr= 5;  
    System.out.println(((Integer)nr).hashCode());  
}
```

Cast to wrapper, to access hashCode()

... or implicitly. This is called autoboxing

```
private static void autoBoxing() {  
    Integer intObject = 3;  
    int intPrimitive = Integer.valueOf(3);  
}
```

Literal 3 primitive is automatically converted to wrapper (autoboxing)

Integer is automatically converted to int (auto-unboxing)

Wrappers: boxing/unboxing



Heads up

Boxing takes (a little) time



Exercise

What's the problem?

```
private static void whatsMyProblem() {  
    Integer a = 5;  
    Integer b = 6;  
    Integer sum = a + b;  
}
```

```
private static void whatsYourProblem() {  
    Integer a = 5;  
    a++;  
}
```

Operators only work on primitives => auto-unboxes, then autoboxes

Only use wrappers when needed!

Don't use them for regular arithmetic → overhead
Use primitive type variables instead, so int instead of Integer!

Predict the output



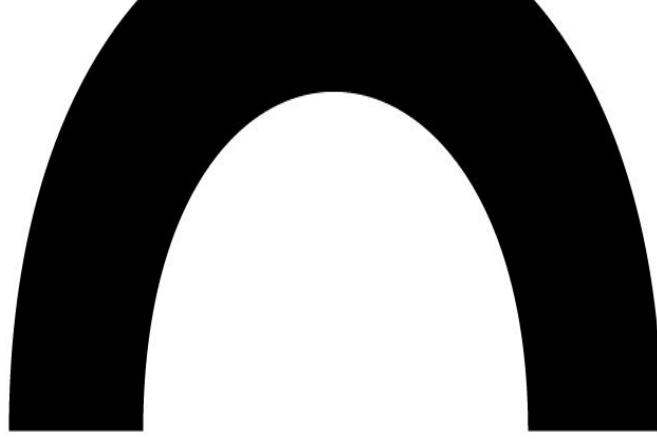
Boxing

```
private static void callMe(long nr) {  
    System.out.println("long wins!");  
}  
  
private static void callMe(Integer nr) {  
    System.out.println("Wrapper wins!");  
}  
  
public static void main(String[] args) {  
    int nr= 5;  
    callMe(nr);  
}
```

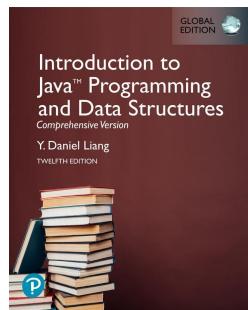
long wins!

– What takes precedence?

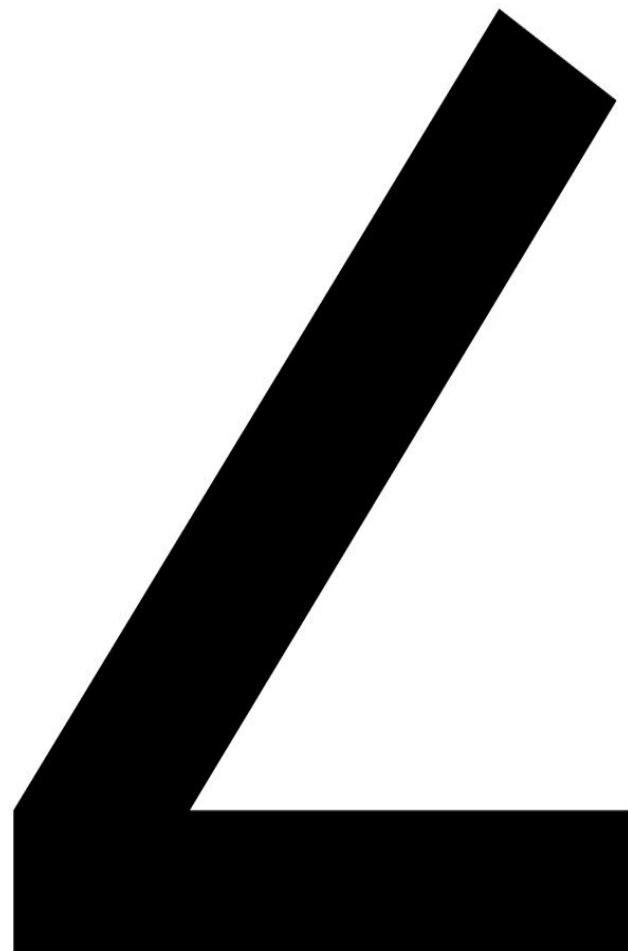
- widen **int** to **long**
- autobox **int** to **Integer**



Collections



Liang: chapters 20 & 21



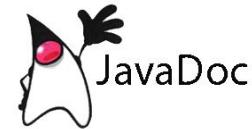
Collections Framework

- **Framework** to handle collections of objects

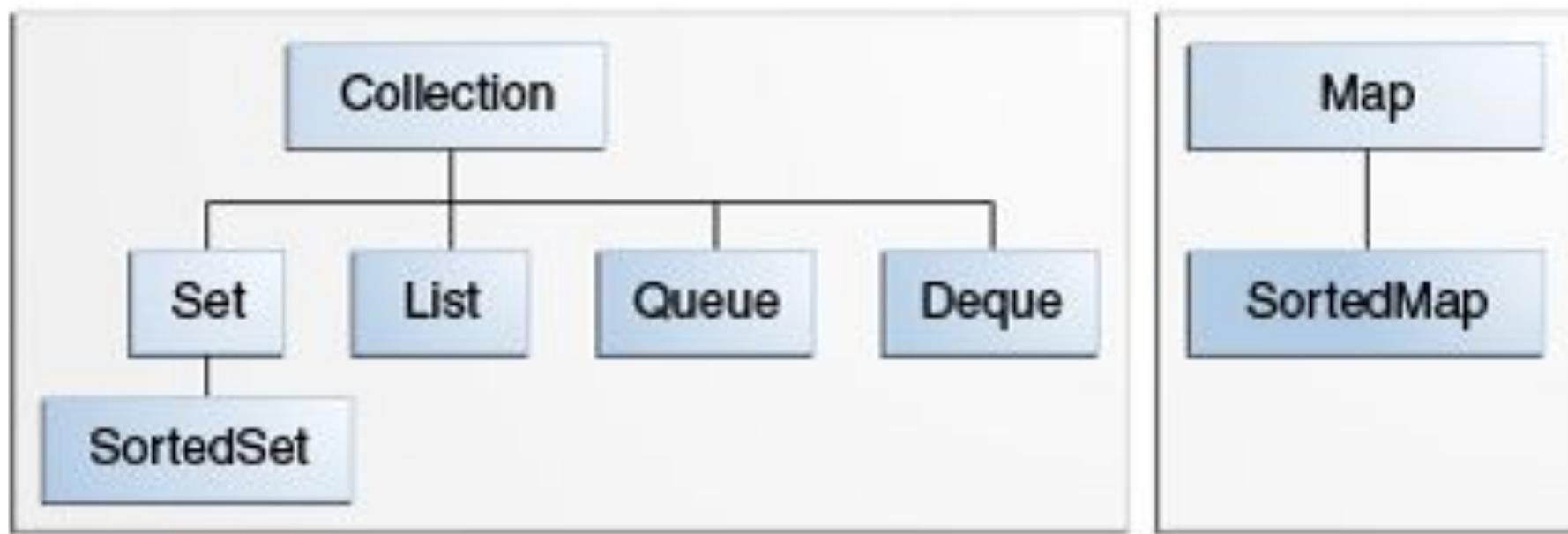
"Don't we have arrays for that?"

- Yes, but:
 - Arrays have a fixed length, Collections can grow.
 - Collections are fully object oriented
 - have methods
 - offer a choice of implementations for each interface
- Collections only work with reference type variables (objects)
→ you can't add variables of a primitive type, say **int**, to a collection. Use an object of type **Integer** instead!
- Arrays advantages: are a little faster and can work with primitive as well as reference type variables (objects)

Collections Framework



- A collection of interfaces and classes
- Defined in package `java.util`
- Major Collection interfaces:



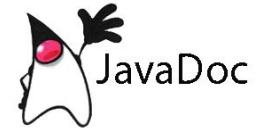
Collection

super interface for all Collections (except Map).

Methods: `add`, `remove`, `clear`, `contains`,
`iterator`, `size`, `toArray`, ...

- Can contain any `reference` type (`Object`)
 - can contain `null`
- Primitive types: first wrap to `Integer`, `Double`, ...

Principal Collection interfaces



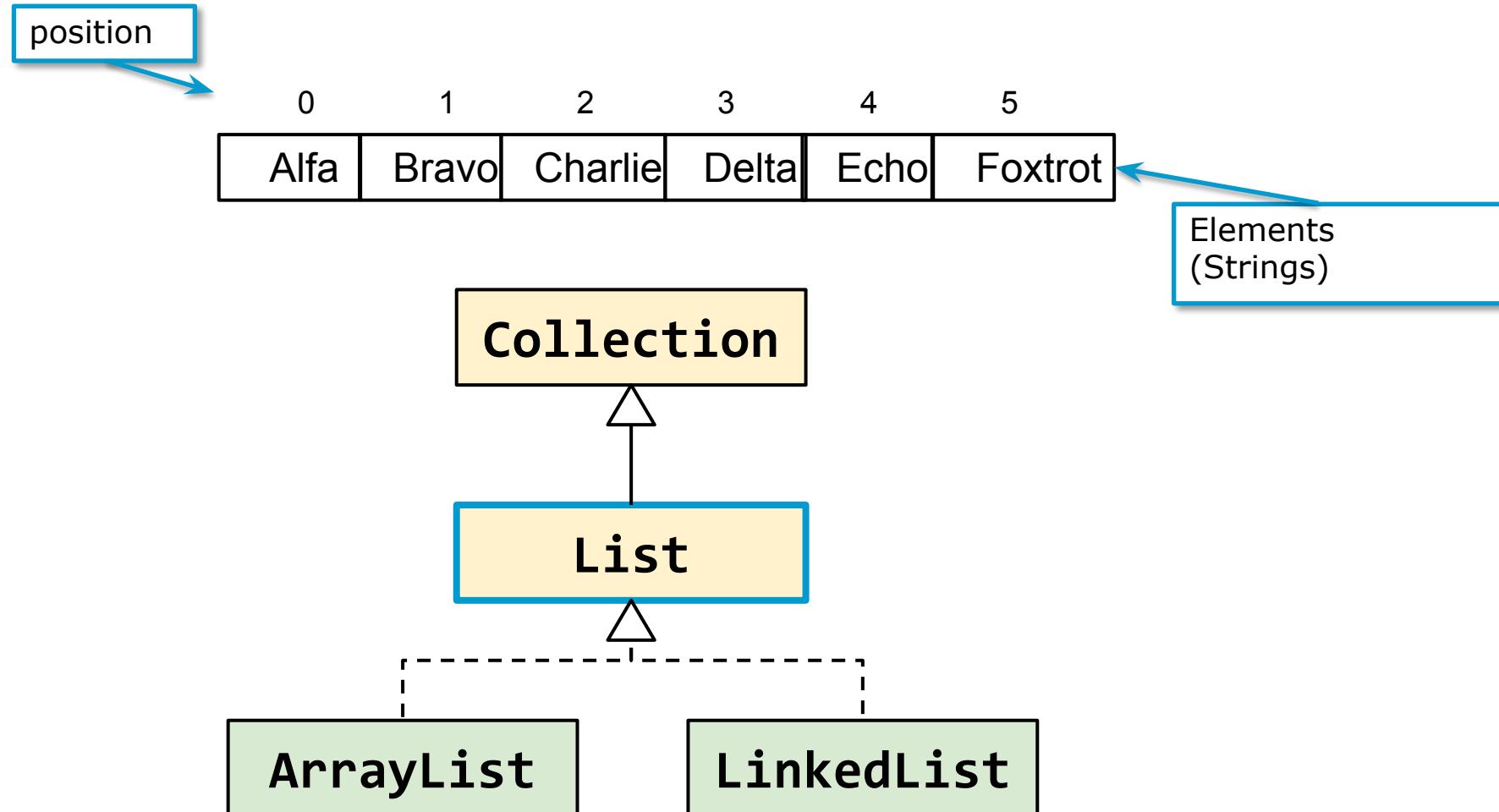
- **List**: sequential collection (order determined by unique number for each element, just like in array)
- **Set**: collection of **unique** objects (no duplicates)
- **Map**: Interface for *key-value* pair Collections
- **SortedSet** and **SortedMap** have additional methods for sorting

List



List interface

- **get/set** □ get/set element at given position
 - 0 based (like array)



ArrayList

- Concrete implementation of the [List](#) interface
- Works like a dynamically growing array
 - **size** (method inherited from Collection): current number of elements



Fast lookup by unique index



Slower for adding/removing elements than other collections

```
List myList = new ArrayList();
```

variable can use interface type:
List (recommended)

new always takes a concrete
class: ArrayList

<Generics>

- Collections accept objects of any class (heterogeneous)
- Often we want homogeneous collections
 - Arrays are homogeneous
- Generics allow for homogeneous collections
 - Add a "generic type argument" to a **Collection** between <angled brackets>
- Recommended syntax:

reads: List OF String

compiler can derive generic type from declaration so you **may** leave it empty (diamond operator)

```
List<String> myList = new ArrayList<>();
```

- myList will only accept Objects of type String now



Predict the output (1)

ArrayList1

```
List<String> myStringList = new ArrayList<>();  
  
myStringList.add("Alfa");  
myStringList.add("Bravo");  
myStringList.add("Charlie");  
myStringList.add("Delta");  
  
for (String element : myStringList) {  
    System.out.print(element + " ");  
}  
  
System.out.println(myStringList.size());
```



Predict the output (2)

ArrayList2

```
List<String> myStringList = new ArrayList<>();  
myStringList.add(0, "Alfa");  
myStringList.add(0, "Bravo");  
myStringList.add(0, "Charlie");  
myStringList.add(0, "Delta");  
  
for (int i = 0; i < myStringList.size(); i++) {  
    System.out.print(myStringList.get(i) + " ");  
}  
  
System.out.println(myStringList.size());
```

List **add** can
have an index

Iterator<E>

- The `iterator()` method can get an Iterator for a collection
 - Yet another way to loop over the collection
- Iterator methods:
 - `boolean hasNext()` → returns true if there are any elements left.
 - `E next()` → Moves to next element and returns it.
 - `void remove()` → Removes the current element from the collection.



Guide: when using a for loop on a collection it is unsafe to remove its elements (it is like removing the steps from the stairs you are climbing). If you want to remove elements while looping over a collection always use the `Iterator remove`-method!

Iterator<E>

ArrayList3

```
for(Iterator<String> it=myList.iterator(); it.hasNext();){  
    System.out.println(it.next());  
}
```

Using <generics>
with Iterator

- List also has the `listIterator()` method (other Collections don't)
 - Additional ListIterator methods:
 - `boolean hasPrevious()` → returns true if there is a preceding element
 - `E previous()` → Moves to previous element and returns it.
 - `void set (E element)` → replaces the current object with element



Exercise

Complete exercise Java 1.01

Array ⇔ List<E> (1)

- To convert an array to a List and vice versa...

ListCopy1

```
String[] myArray = { "Alfa", "Bravo", "Charlie", "Delta" };  
List<String> myList = new ArrayList<>();
```

```
for (String name : myArray) {  
    myList.add(name);  
}
```

```
System.out.println("List => Array: " + myList);
```

```
System.out.print("Array => List:");  
String [] copyArray = new String [myList.size()];  
for (int i=0;i<myList.size();i++){  
    copyArray[i] = myList.get(i);  
    System.out.print(" " + copyArray[i]);  
}
```

You can directly print a List using its **toString** method!

*List => Array: [Alfa, Bravo, Charlie, Delta]
Array => List: Alfa Bravo Charlie Delta*

Array ⇒ List<E> (2)

... but there are easier ways

- **Arrays** is a class with static utility methods for arrays
 - Take a look! `toString`, `sort`, `copyOf`, `equals`...
- **Arrays.asList** returns a **List** that is a **fixed-size** view on the array

ListCopy2

```
String[] myArray = { "Alfa", "Bravo", "Charlie", "Delta" };  
List<String> myList = Arrays.asList(myArray);  
  
System.out.println(myList);
```

[Alfa, Bravo, Charlie, Delta]



fixed-size view: `myList.add("Echo")`;
throws an "**UnsupportedOperationException**" at runtime

Array ← List<E> (3)

... but there are easier ways

- List **toArray** method:

```
List<String> myList = new ArrayList<>();  
myList.add("Alfa");  
myList.add("Bravo");  
myList.add("Charlie");  
myList.add("Delta");  
String[] myArray = myList.toArray(new String[0]);  
for (String word : myArray) {  
    System.out.print(word + " ");  
}
```

toArray parameter **must** be an empty **array** of the desired type. (simply casting will not work!)



Alfa Bravo Charlie Delta

Intermezzo: List.of

- The static method `of` in `List` lets you directly create a small **read-only** `List`
⚠ **read-only:** you cannot change any element in the `List`
- Same example with `List.of`

ListCopy3

```
List<String> myList =  
    List.of("Alfa", "Bravo", "Charlie", "Delta");  
  
String[] myArray = myList.toArray(new String[0]);  
for (String word : myArray) {  
    System.out.print(word + " ");  
}
```

Alfa Bravo Charlie Delta

List<E> ⇒ List<E> (4)

How to turn a restricted list (fixed size or read only) into a normal list?

ArrayList has a constructor that takes another Collection as an argument , and copies it.

ListCopy4

```
List<String> myList = List.of("Alfa", "Bravo", "Charlie", "Delta") ;  
  
// UnsupportedOperationException  
// myList.set(1, "Beta");  
List<String> fullAccessList = new ArrayList<>(myList) ;  
fullAccessList.set(1, "Beta") ;  
  
System.out.println(fullAccessList) ;
```

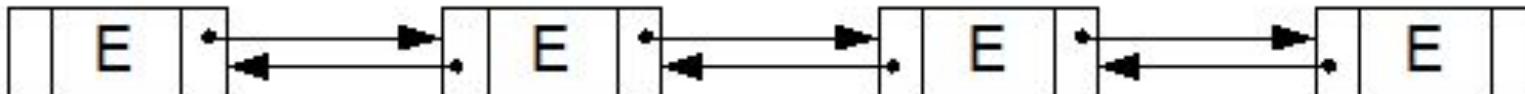
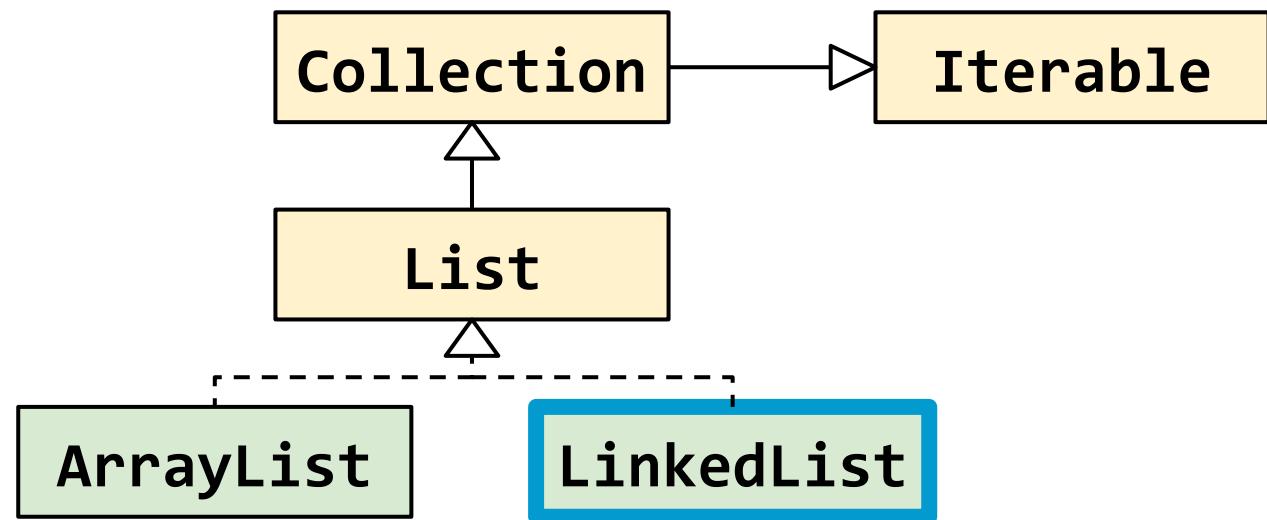
[Alfa, Beta, Charlie, Delta]

LinkedList<E>

- Another implementation of the List interface
- Is a **doubly linked** list: Each element refers to its predecessor and its successor

⊕ Fast for add/remove

⊖ Slower for indexed access



LinkedList<E>

- All methods inherited from `List` and `Collection` interfaces are **identical** to `ArrayList`
- Some extra methods in `LinkedList`:
 - `addLast`/`addFirst`
 - `removeLast`/`removeFirst`
 - ...
- The example on the next slide also shows some additional useful `List` methods (for all types of lists)

LinkedList<E> example

LinkedList

```
List<String> colourList =  
    new LinkedList<>(List.of( "green", "blue", "purple", "red"));  
List<String> grayList =  
    new LinkedList<>(List.of("black", "white", "gray", "silver"));  
colourList.addAll(grayList);
```

```
System.out.println("The colours: " + colourList);  
colourList.subList(4, 6).clear();  
(LinkedList<String>) colourList.addFirst("yellow");  
System.out.println("Changed colours: " + colourList);
```

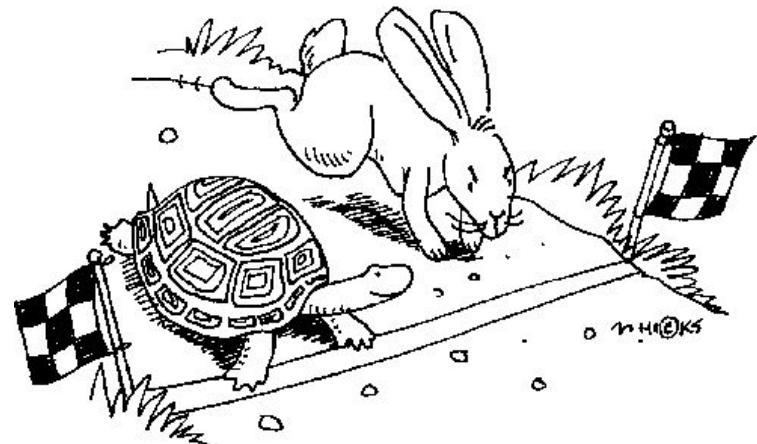
```
ListIterator<String> iterator =  
colourList.listIterator(colourList.size());  
System.out.print("Colours in reverse: ");  
while (iterator.hasPrevious()) {  
    System.out.print(iterator.previous() + " ");  
}
```

LinkedList specific
method: need to cast
List variable

```
The colours: [green, blue, purple, red, black, white, gray, silver]  
Changed colours: [yellow, green, blue, purple, red, gray, silver]  
Colours in reverse: silver gray red purple blue green yellow
```

ArrayList<E> versus LinkedList<E>

- **ArrayList:** **read mostly**
 - fast access by position(get, set)
 - slow on add, remove
- **LinkedList:**
 - fast add and remove
 - fast sequential (and reverse) access
 - slow access by position(get, set)



Study the SpeedTest example code and run it!

Exercises

- Ex 01.01: List names
- Ex 01.02: List even numbers
- Ex 01.03: List drinks

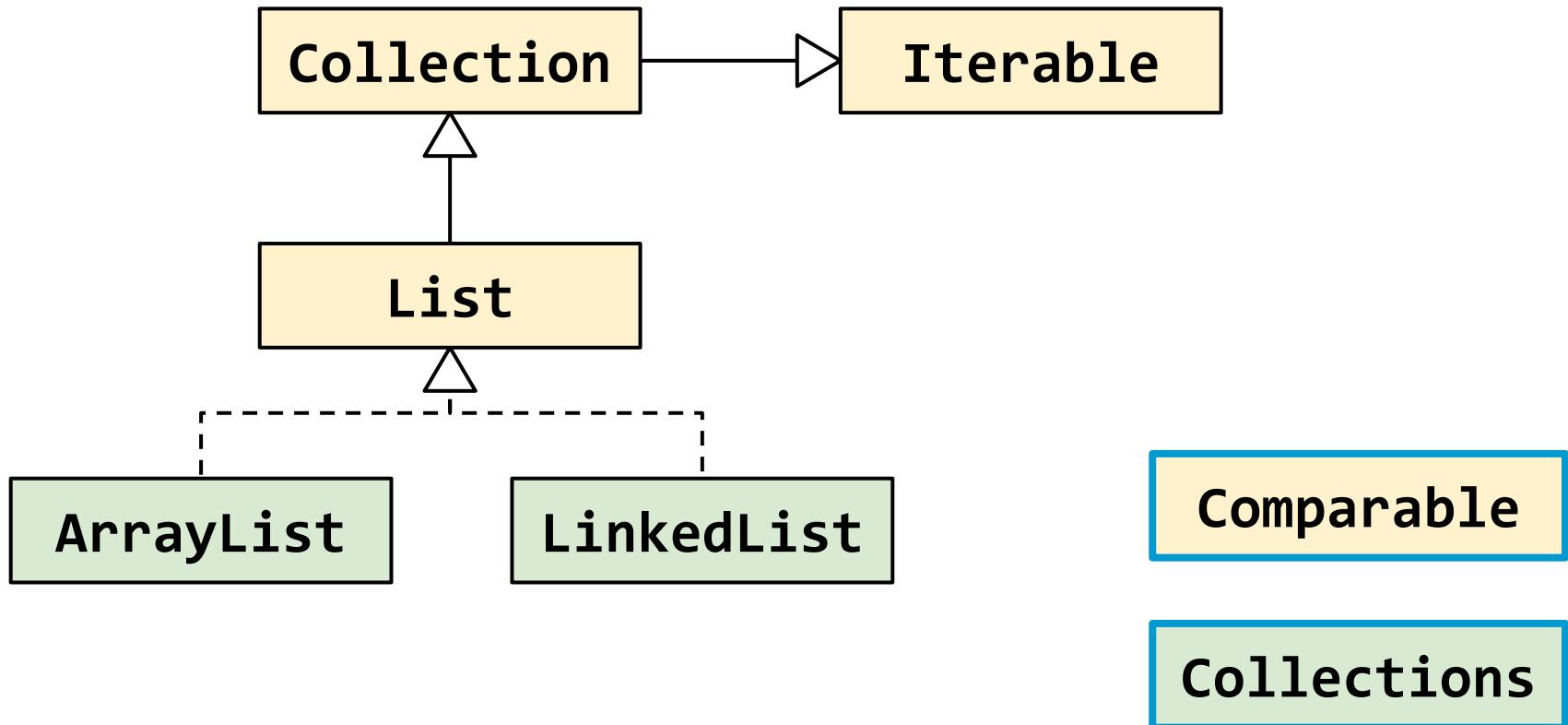




Collections and Sorting



Collections Framework (not complete)



Collections class

- Just like `Arrays`, `Collections` is a class with static utility methods for working with Collections
- Some methods
 - `sort`
 - `binarySearch`
 - `reverse`
 - `shuffle`
 - `min`
 - `max`
 - ...

Collections.sort

`Collections.sort` orders the elements in a `List`.

- `List elements` must implement the `Comparable` interface

```
interface Comparable<T>() {  
    int compareTo(T t);  
}
```

- Most basic Java API classes already implement this interface:
`String, Integer, Double...`

Comparable

```
public static void main(String[] args) {  
    List<String> scrambledDictionary = new ArrayList<>(List.of(  
        "The", "quick", "brown", "fox",  
        "jumps", "over", "the", "lazy", "dog"));  
    Collections.sort(scrambledDictionary);  
    System.out.println(scrambledDictionary);  
}
```

[The, brown, dog, fox, jumps, lazy, over, quick, the]

How to sort Collection<YourClass>?

- PROBLEM: **YourClass** is not Comparable!
- SOLUTION: Let **YourClass** implement interface
Comparable <YourClass>
 - write the **compareTo(Yourclass other)** method in **YourClass**
in compareTo you have to compare **this** object to another object (of the same class)

```
public int compareTo(Student st) {  
    if(this.age < st.age) return -1;  
    if(this.age > st.age) return 1;  
    return 0;  
}
```

Case 1: sort on a Comparable<T> attribute

- Example: you want to sort `List<Driver>` drivers on its String name attribute
 - Have Driver implement `Comparable<Driver>`
 - Add the `compareTo (Driver other)` method to Driver
 - name is a String and String is Comparable, so it has a `compareTo` method
 - in `compareTo(Driver other)` compare `this` Driver to the `other` Driver by comparing the name attributes
`name.compareTo(other.name)`
 - Now you can `Collections.sort(drivers)`

Case 1: sort on a Comparable<T> attribute

```
public class Driver implements Comparable<Driver> {  
    private String name;  
    private int number;  
  
    public Driver(String name, int number) {  
        this.name = naam;  
        this.number = nummer;  
    }  
    // ...  
    @Override  
    public int compareTo(Driver other) {  
        return this.name.compareTo(other.name);  
    }  
    // ...  
}
```

Comparable1

sort on name!

```
public static void main(String[] args) {  
    List<Driver> drivers = new ArrayList<>();  
    drivers.add(new Driver("Kevin", 20));  
    // ...  
  
    Collections.sort(drivers);  
    // ...  
}
```

will sort using compareTo method in Driver.

Case 2: sort on something else

- Example: you want to sort `List<Movie>` movies on its int year attribute
 - Have `Movie` implement `Comparable<Movie>`
 - Add the `compareTo (Movie other)` method to `Movie`
 - year is an int primitive
 - in `compareTo (Movie other)`
 - return a negative number if `this` Movie year is before (smaller than) the `other` Movie year
 - return 0 if `this` Movie year equals the `other` Movie year
 - return a positive number if `this` Movie year is after the `other` Movie year
 - => `return this.year - other.year`
 - For other sorting orders, follow the same convention: return a negative number if `this` is smaller than `other`, 0 for equal,...
 - Now you can `Collections.sort(movies)`

Case 2: sort on something else

```
public class Movie implements Comparable<Movie> {  
    private String title;  
    private int year;  
  
    public Film(String title, int year) {  
        this.title = title;  
        this.year = year;  
    }  
    // ...  
    @Override  
    public int compareTo(Movie other) {  
        return year - other.year;  
    }  
    // ...  
}
```

Comparable2

sort on year!

```
public static void main(String[] args) {  
    List<Movie> movies = new ArrayList<>();  
    movies .add(new Movie("The Godfather", 1972));  
    // ...  
    Collections.sort(movies);  
    // ...  
}
```

will sort using compareTo method in Movie.

More static Collections utility methods

- **shuffle** reorders elements of a [List](#) randomly.
- **reverse** reverses order of elements in a [List](#).
- **min / max** returns largest/smallest element in a [Collection](#)
- **binarySearch** looks up an element in a [sorted List](#):
 - returns position if found, a negative number if not found
- **addAll** adds all elements specified as parameters (or an array) to a Collection

Collections example

Collections1

```
List<String> fruitList = new ArrayList<>(
    List.of("banana", "pear", "apple", "prune", "coconut")) ;
Collections.sort(fruitList);
System.out.println("sorted: " + fruitList);
Collections.shuffle(fruitList);
System.out.println("shuffled: " + fruitList);
Collections.reverse(fruitList);
System.out.println("reversed: " + fruitList);
System.out.println("min: " + Collections.min(fruitList));
System.out.println("max: " + Collections.max(fruitList));
Collections.sort(fruitList);
System.out.println("banana at: " +
    Collections.binarySearch(fruitList, "banana"));
System.out.println("cucumber at: " +
    Collections.binarySearch(fruitList, "cucumber"));
```

sort before
binarySearch

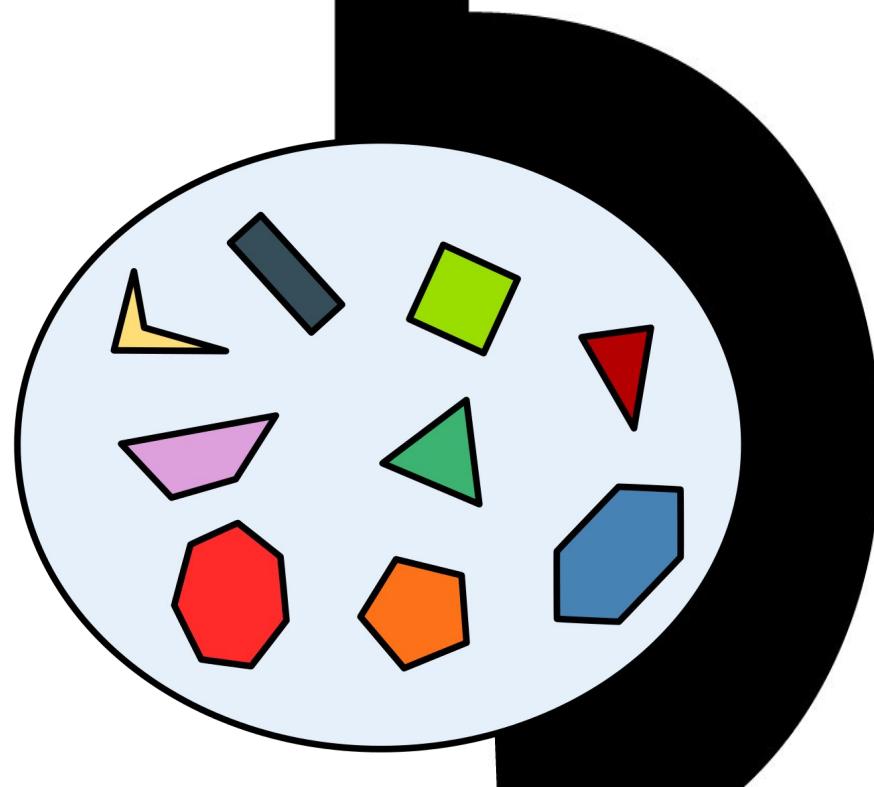
```
sorted: [apple, banana, coconut, pear, prune]
shuffled: [pear, apple, coconut, banana, prune]
reversed: [prune, banana, coconut, apple, pear]
min: apple
max: prune
banana at: 1
cucumber at: -4
```

Exercises

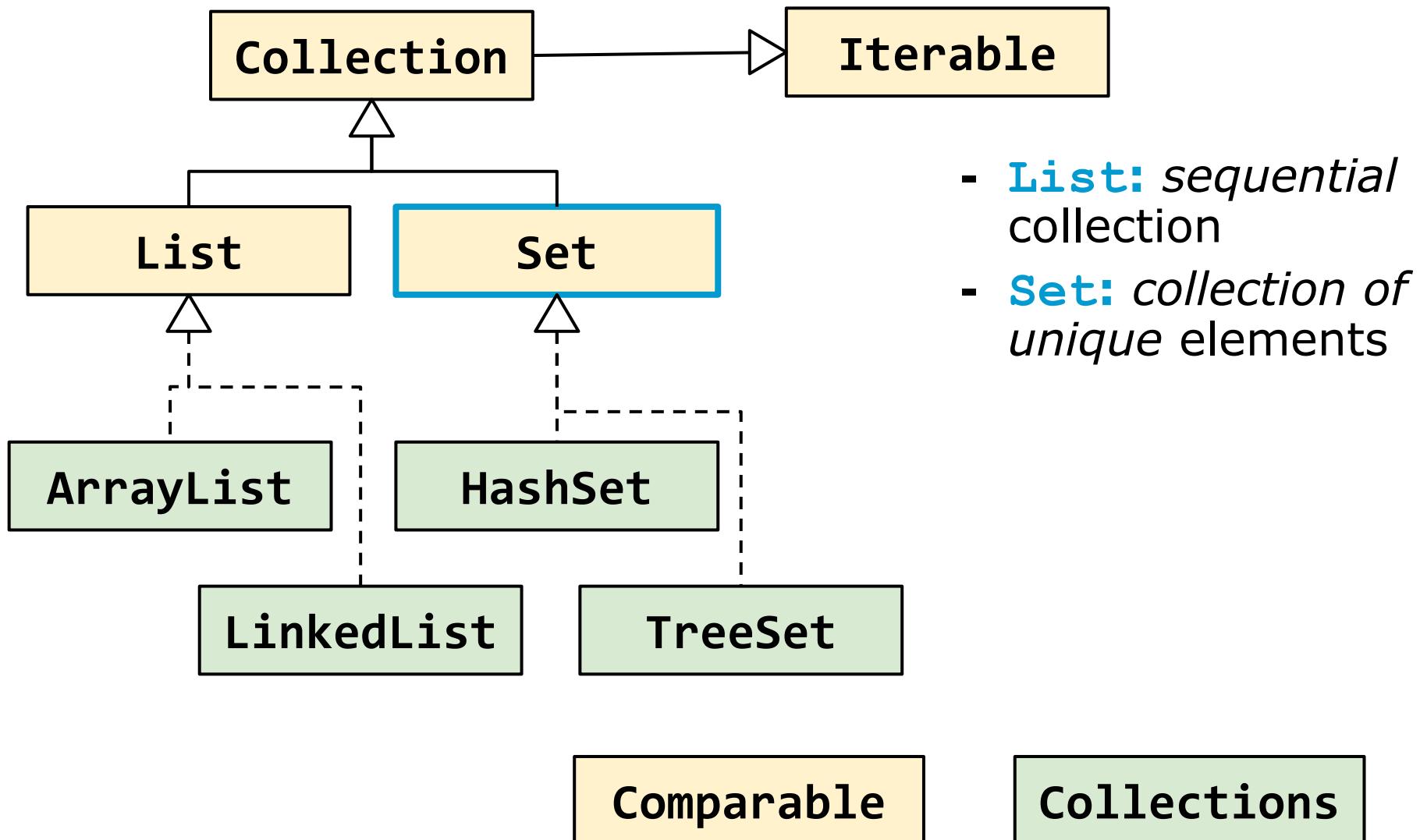
- Ex 01.04: List drinks V2
- Ex 01.05: List even numbers V2
- Ex 01.06: List Brands
- Ex 01.07: Scores



Set



Collections Framework (not complete)



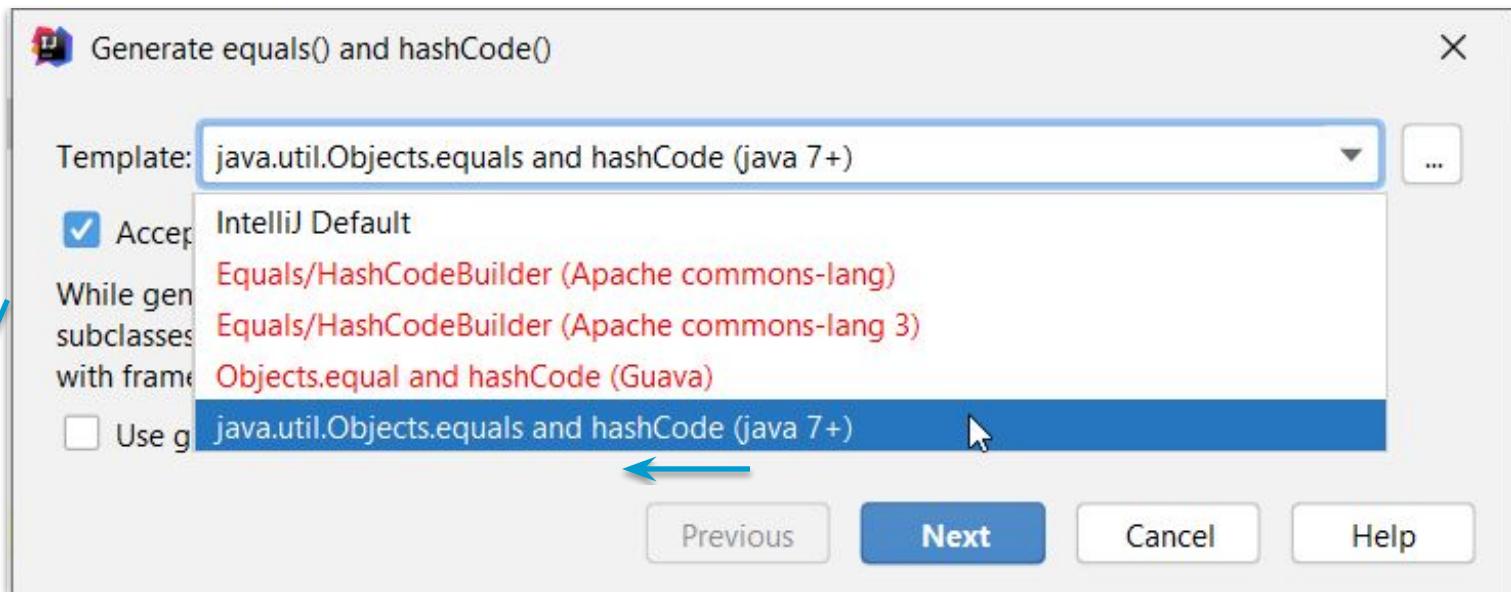


What is a hashCode?

- A hashCode is like a checksum
- Method in Object: `int hashCode()`
 - hashCode is a number, characteristic for an object, but not unique!
 - for objects that are equal (`equals()` returns `true`), `hashCode()` **must** return the same number
 - if `hashCode()` for 2 objects returns a different number, they are likely different, but can be the same
 - The default implementation likely returns different numbers for objects with different references, even if they are equal

What is a hashCode?

- If you override equals, you must override hashCode to return a number that is only based on the attributes used in equals



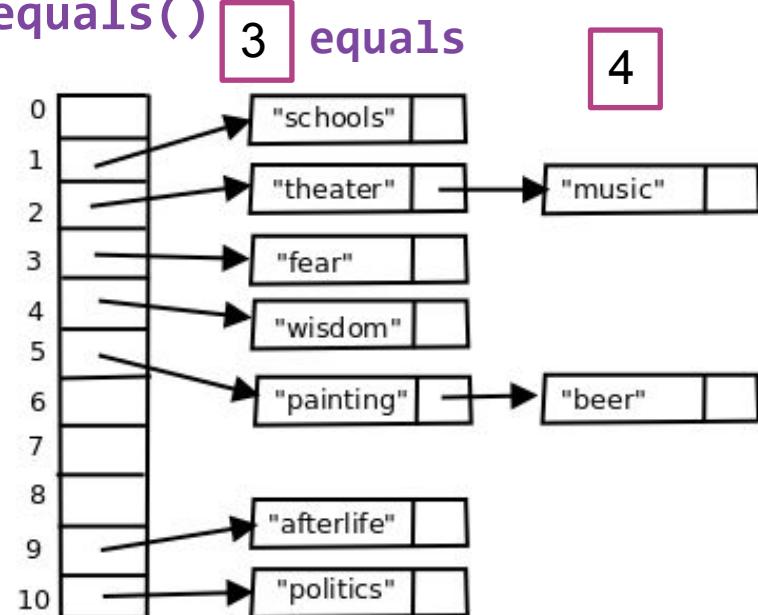
IntelliJ: ALT+INSERT will generate both equals() and hashCode()

How does a HashSet<E> work?

- When adding an element, internally the HashSet:
 - hashCode** method on the object called which returns a number (hashcode)
 - used the hashcode to determine the index in the underlying hashtable
 - At position index there is a 'bucket': a (small) linked list of elements
 - all the elements at the hashcode index are compared with the new element using **equals()**
 - If the element is not present it is added to the bucket

1
hashCode method generates a hashCode

2
hashcode % capacity \Rightarrow index



How does a HashSet<E> work?

For your information (will not be asked on the exam)

When adding an element, internally the HashSet uses the following formula to determine the index in the underlying hashtable (default `capacity` = 10)

- a. when `hashcode <= 65535`

```
    index = hashcode % capacity
```

or

```
    index = hashcode & (capacity - 1)
```

- b. when `hashcode > 65535`

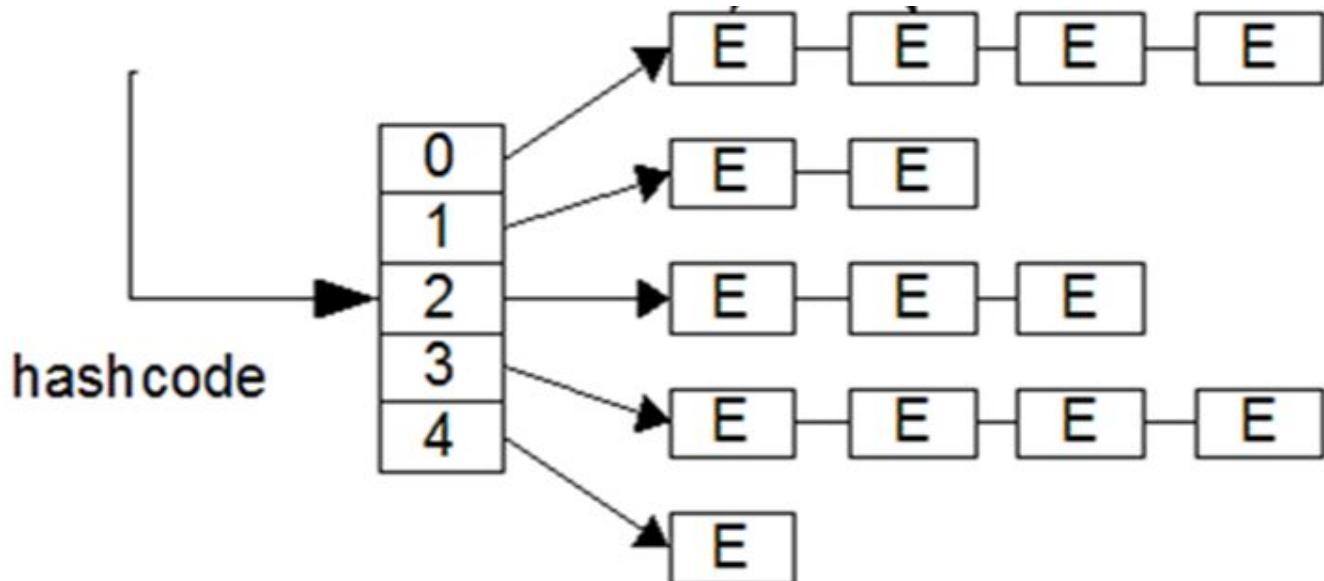
```
    index = (hashCode ^ (hashCode >>>16))  
          & (capacity - 1)
```

How does a HashSet<E> work?

- Combines advantages of **ArrayList** (fast lookup using the hashCode) + **LinkedList** (fast add and remove)

 **Lookup, add and remove are fast**

 **Not ordered**



HashSet<E> : example 1

HashSet1

```
public static void main(String[] args) {  
    Set<String> set = new HashSet<>();  
    Collections.addAll(set,  
        "red", "red", "white", "blue", "green",  
        "gray", "white", "orange", "green");  
  
    System.out.printf("%d colours: %s\n", set.size(), set);  
}
```

6 colours: [red, orange, green, gray,
white, blue]

Remark: Order has changed
and no duplicates!

HashSet<E> : example 2

HashSet2

```
public class Driver {  
    private String name;  
    private int number;  
  
    public Driver(String name, int number) {  
        this.name = name;  
        this.number = number;  
    }  
  
    @Override  
    public int hashCode() {  
        return number;  
    }  
  
    @Override  
    public String toString() {  
        return String.format("%-10s %2d", name, number);  
    }  
}
```

BAD PRACTICE

hashCode and no **equals** method



HashSet<E> : example 2

```
public class HashSetDemo2 {  
    public static void main(String[] args) {  
        Driver[]  
        Set<Driver> drivers = new HashSet<>();  
        drivers.add(new Driver("Kevin", 20));  
        drivers.add(new Driver("Carlos", 55));  
        drivers.add(new Driver("Nico", 6));  
        drivers.add(new Driver("Valtteri", 6));  
  
        System.out.printf("%d drivers: %s", drivers.size(), drivers);  
    }  
}
```

Duplicate
number!

4 drivers: [Kevin 20, Nico
Valtteri 6, Carlos 55]

Duplicates go undetected
because overridden
hashCode is not consistent
with inherited **equals**
method.



HashSet<E> : example 3

HashSet3

```
public class Driver {  
    private String name;  
    private int number;  
  
    public Driver(String name, int number) {  
        this.name = name;  
        this.number = number;  
    }  
  
    @Override  
    public boolean equals(Object o) {  
        return number==((Driver) o).number;  
    }  
  
    @Override  
    public String toString() {  
        return String.format("%-10s %2d", name, number);  
    }  
}
```

4 drivers: [Kevin 20, Nico 6,
Valtteri 6, Carlos 55]

BAD PRACTICE

equals and no **hashcode** method



Duplicates go undetected because overridden **equals** is not consistent with inherited **hashcode** method.

6,

HashSet<E> : example 4

HashSet4

```
public class Driver {  
    private String name;  
    private int number;  
    //...  
    @Override  
    public boolean equals(Object o) {  
        if (this == o) return true;  
        if (!(o instanceof Driver)) return false;  
        return number == ((Driver) o).number;  
    }  
  
    @Override  
    public int hashCode() {  
        return number;  
    }  
  
    @Override  
    public String toString() {  
        return String.format("%-10s %2d", name, number);  
    }  
}
```

GOOD PRACTICE:

Both **equals** and **hashCode** overridden and based on same attribute(s)

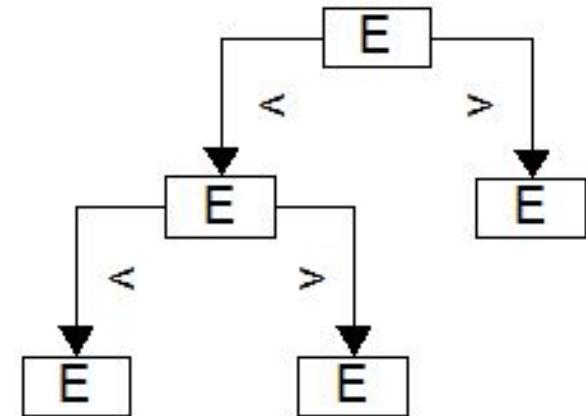


3 drivers: [Kevin 20, Nico 6, Carlos 5]

Duplicate number eliminated!

TreeSet<E>

- Set with hierarchical **treestructure**
- Elements are automatically **sorted**
 - heads up: elements that are ordered equally are considered duplicates (and are removed)
 - Implements **Set** plus **SortedSet** and **NavigableSet**
 - extra methods:
 - **E first()**
 - **E last()**
 - **E ceiling (E element)**
 - **E floor (E element)**
 - ...

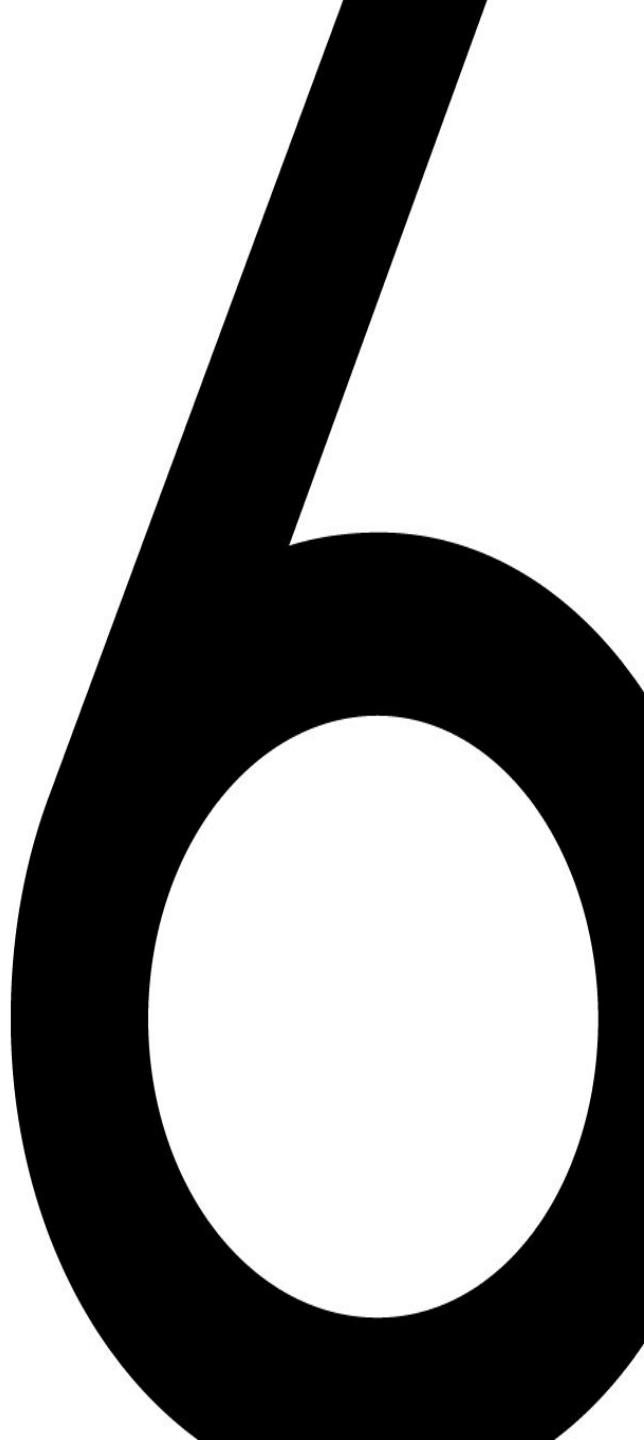


Exercises

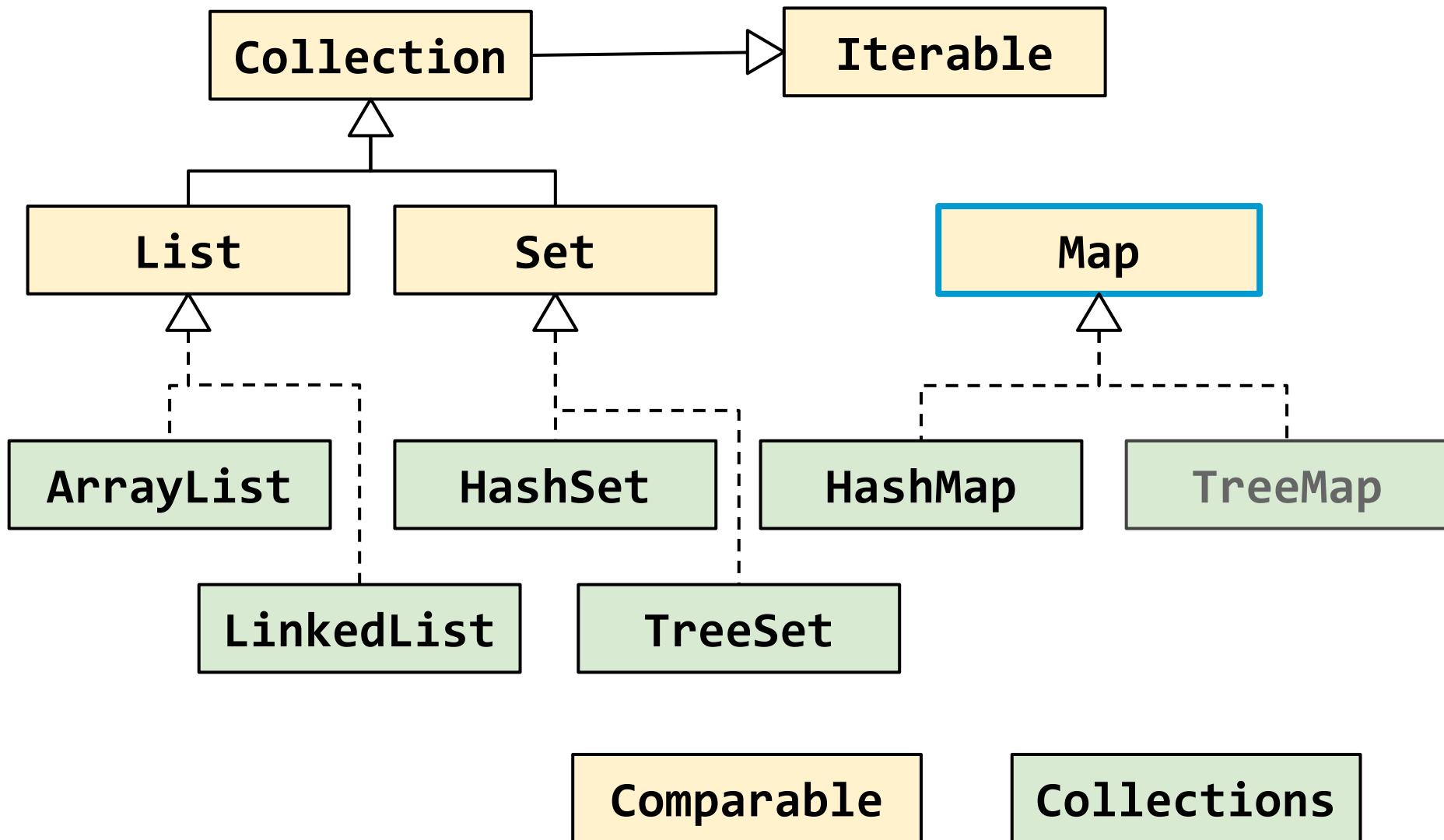
- Ex 01.08: Set of Girls
- Ex 01.09: Set of Girls V2
- Ex 01.10: LottoDraw
- Ex 01.11: Actresses



Map

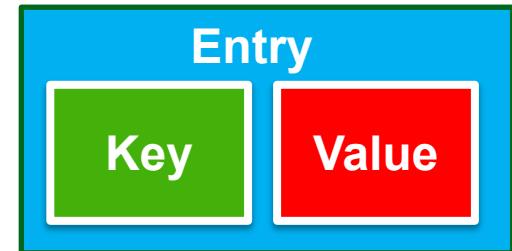


Collections Framework (not complete)



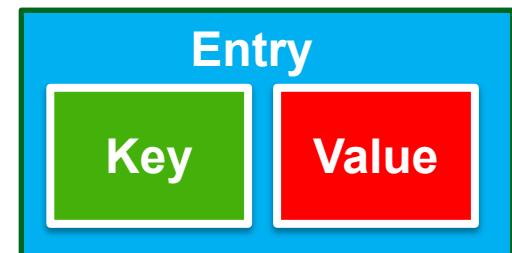
Map<K,V>

- A **Map** stores **Key** / **Value** pairs. A key/value pair is an **Entry**.
 - You can look up a **Value** in a **Map** using the **Key**
 - like you look up an element in a list using its position
 - **Key** can be any Object, not just an int position
 - **Key** in a map is unique (like an int position). If you add a new value object with an existing key, the value object that was present is replaced



Map<K,V> methods

- **V put(K key, V value)**
 - adds an entry
 - if the key exists the previous value is replaced and returned
- **V get(K key)**
 - returns the value associated with the key
- **int size()**
- **boolean containsKey(K key)**
- **boolean isEmpty()**
 - is the map empty?
- **Set<K> keySet()**
 - returns a **Set** containing all keys
- **Collection<V> values()**
 - returns **Collection** containing all values



Some Map<K,V> implementations

- **HashMap<K,V>:**
 - entries are internally stored in a HashSet based on the key hashCode
- **TreeMap<K,V>:**
 - entries are internally stored in a TreeSet
 - entries are **automatically sorted**

HashMap<K,V> WordCounter(1/2)

MapWordCounter

```
public static void main(String[] args) {  
    Map<String, Integer> map = new HashMap<>();  
  
    key = String; value = Integer  
  
    Scanner scanner = new Scanner(System.in);  
    System.out.print("Enter a sentence: ");  
    String sentence = scanner.nextLine();  
    String[] words = sentence.split(" ");  
  
    for (String string : words) {  
        String word = string.toLowerCase();  
        if (map.containsKey(word)) {  
            int amount = map.get(word);  
            map.put(word, amount + 1);  
        } else {  
            map.put(word, 1);  
        }  
    }  
}
```

The words are the **keys** in the **map**. The numbers are the **values**. **containsKey** checks if the **key** is already present

If key is present: value is retrieved, incremented and overwritten

If key is not present: the key is added with initial value: **1**

HashMap<K,V> WordCounter(2/2)

```
System.out.printf("\nMap contains:\n%-10s %10s\n",
                  "Key", "Value");
for (String key : map.keySet()) { ← loop over keys
    System.out.printf("%-10s %10d\n", key, map.get(key));
}
System.out.printf("\nsize: %d\nisEmpty: %b\n",
                  map.size(), map.isEmpty());
} // main()
```

get **value** (number of occurrences) for **key** (word)

Enter a sentence: **To be or not to be**

Map contains:

Key	Value
not	1
be	2
or	1
to	2

size: 4

isEmpty: false

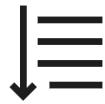


Exercises

- Ex 01.12: Map of Girls
- Ex 01.13: Map of Drivers (Extra)

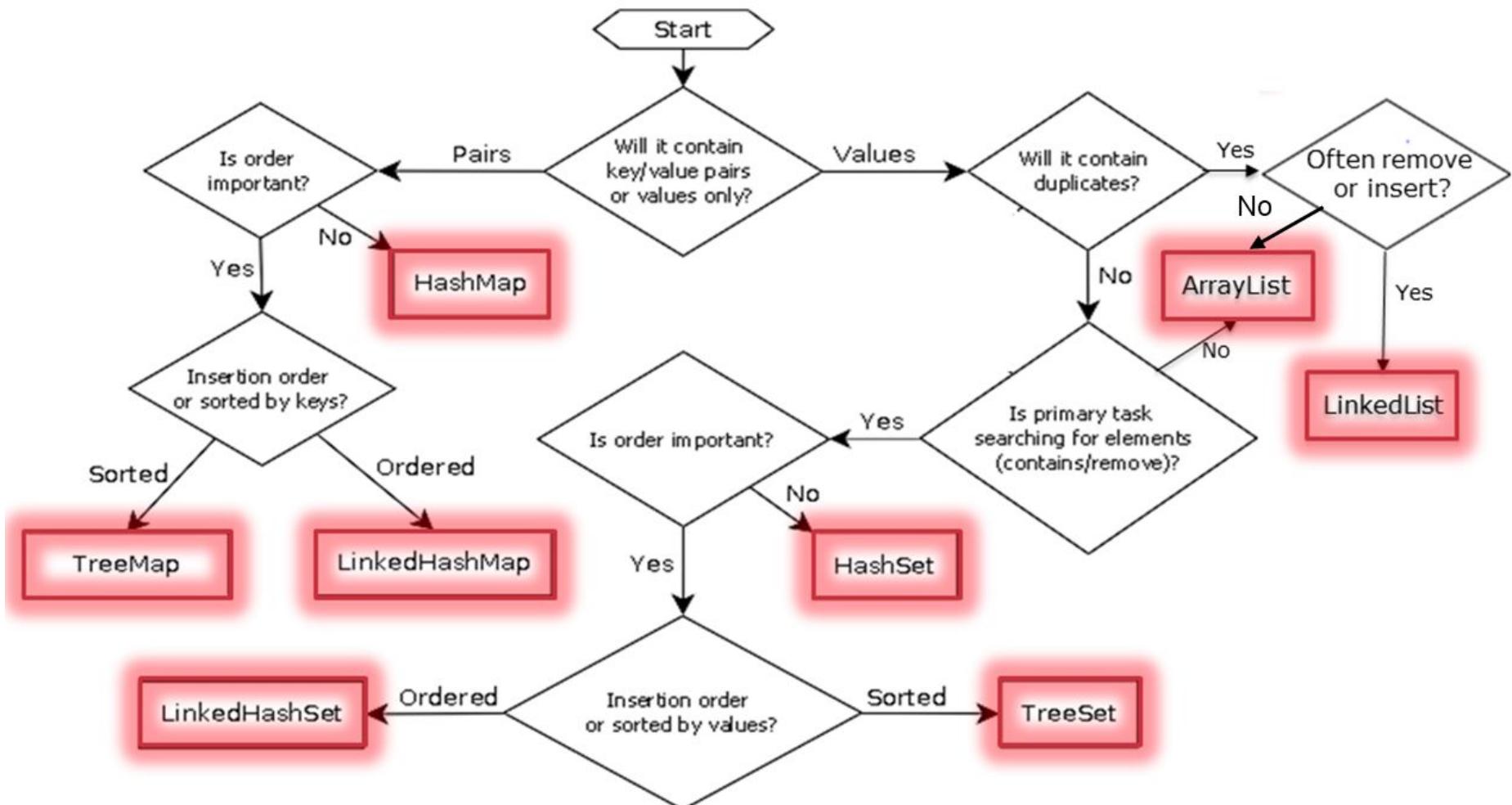


Summary: interfaces and implementations



- **array**: elements accessible by position (=ordered), duplicates allowed. No methods (not object oriented)
- **List**: elements accessible by position, duplicates allowed
 - **ArrayList**: efficient when data is mostly read
 - **LinkedList**: efficient when data is mostly written
- **Set**: no duplicates
 - **HashSet**: not ordered (uses equals and hashCode)
 - **LinkedHashSet**: ordered (uses equals and hashCode)
 - **TreeSet**: sorted (doesn't use equals and hashCode!
uses comparison algorithm instead!)
- **Map**: key – value pairs (no duplicate keys)
 - **HashMap**: not ordered
 - **LinkedHashMap**: ordered
 - **TreeMap**: sorted (using key)

Choose the best implementation





Test

Choose the best implementation

- unique elements, automatically sorted
- duplicate elements, frequent insert/remove
- non unique values, accessible by key
- non unique values, accessible by index
- Key-value pairs, sorted by key

TreeSet**LinkedList****HashMap****ArrayList****TreeMap**

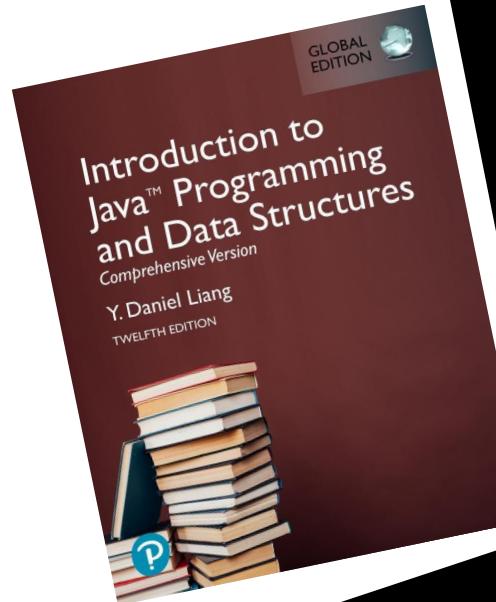


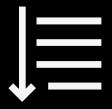
1. Wrappers
2. Collection
3. List
4. Collections and Sorting
5. Set
6. Map

enums - Inner classes - Exceptions - JavaFX

Programming 2: Java

KdG University of Applied
Sciences and Arts





Agenda

- enums
- Inner classes
 - Nested classes
 - Static nested classes
- Exceptions
 - Types of exceptions
 - Throwing Exceptions
 - Custom exceptions
- JavaFX setup



Appendix I



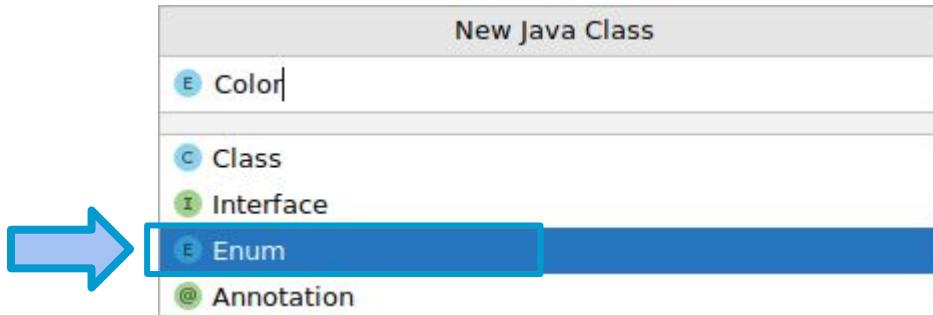
enums

enums: definition

- Example:

```
public enum Color {  
    BLACK, WHITE, RED, GREEN, BLUE, YELLOW;  
}
```

- Creating one in IntelliJ:



UML representation

enums: definition

- Enumerations: **enum**
 - Custom type that holds a collection of constants
 - Keeps code readable and maintainable
 - Use keyword **enum** instead of **class**
- Convention: write constant values using **ALL_CAPITALS**



```
public enum Size {  
    S, M, L, XL, XXL;  
}
```

```
public enum Gender {  
    MALE, FEMALE, X;  
}
```

- Each value has a unique index within the enum type, which is called the **ordinal**

enums: application

```
public enum Color {  
    BLACK, WHITE, RED, GREEN, BLUE, YELLOW  
}
```

M2.1_enums color.ColorDemo

```
public class ColorDemo {  
    public static void main(String[] args) {  
        Color foreground = Color.BLUE;  
        Color background = Color.WHITE;  
        printColor(foreground);  
        printColor(background);  
    }  
  
    private static void printColor(Color color) {  
        System.out.println("Name: " + color.name());  
        System.out.println("Ordinal: " + color.ordinal());  
    }  
}
```

Name: BLUE
Ordinal: 4
Name: WHITE
Ordinal: 1

Assigns enum constant values to enum variables.

`name()` returns the name of the value.

`ordinal()` returns the 'index' of the constant value within the enum type.

enums: application

- Checking for a specific value
 - Using **if** or using a **switch**:

```
M2.1 enums color.ColorDemo
```

```
private static void printColor2(Color color){  
    if (color == Color.BLACK){  
        System.out.println("Black is beautiful!");  
    }  
  
    System.out.println( switch(color){  
        case BLACK -> "Black";  
        case WHITE ->"White";  
        case RED -> "Red";  
        case GREEN -> "Green";  
        case BLUE -> "Blue";  
        case YELLOW -> "Yellow";  
    });  
}
```



enums: application

- Printing each possible enum value:
 - method **values()**

```
M2.1_enums color.ColorDemo
```

```
private static void printAllColorEnumValues() {  
    for(Color color : Color.values()) {  
        System.out.printf("%nName: %s (%d)",  
                          color.toString(),  
                          color.ordinal());  
    }  
}
```

```
Name: BLACK (0)  
Name: WHITE (1)  
Name: RED (2)  
Name: GREEN (3)  
Name: BLUE (4)  
Name: YELLOW (5)
```

- by default, **toString** returns the same value as **name**

Extending enums

- enum has a lot in common with a class
 - Can have attributes, constructors, as well as methods
 - It inherits from `Object`
- Example:
 - Add an RGB property for each color
 - Pass the RGB-value as a parameter to the constructor
 - `toString` could show the name as well as the RGB-value
- An enum constructor is implicitly private! Why would that be the case?
 - Adding additional constant values from outside of the enum (and at runtime) is not allowed!
 - Better not to add setters either!

Extending enums

```
M2.1_enums color_extend.Color
```

```
public enum Color {  
    BLACK(0x000000), WHITE(0xFFFFFFFF),  
    RED(0xFF0000), GREEN(0x00FF00),  
    BLUE(0x0000FF), YELLOW(0xFFFF00);  
  
    private final int rgb;  
  
    Color(int rgb){  
        this.rgb = rgb;  
    }  
  
    public int getRgb() {  
        return rgb;  
    }  
  
    @Override  
    public String toString() {  
        return String.format("%s (%x)", name(), rgb);  
    }  
}
```

The constructor is called for each of the enum's constant values. We're passing the RGB-value. Notice the absence of 'new'!

Custom private attribute with an accessor.

Constructor to pass the RGB-value: always (automatically) private

We're showing the name as well as the hexadecimal representation (%x) of the RGB attribute.

import static

import static allows you to
use a static without the className

```
M2.1_enums color_extend.ColorApp
```

```
import static be.kdg.java.enums.Color.WHITE;

class ColorApp {
    public static void main(String[] args) {
        Color color = WHITE; //Color.WHITE not needed
        System.out.println(color);
    }
}
```

WHITE (fffff)



Nesting enums

- It's possible to declare an enum within another class (→ nested class or inner class; see **section 2**)

```
public class Car {  
    public enum Paint {  
        SOLID, METALLIC, PEARL, MATTE  
    }  
  
    private String brand;  
    private Color color;  
    private Paint paint;  
  
    public Car(String brand, Color color, Paint paint) {  
        this.brand = brand;  
        this.color = color;  
        this.paint = paint;  
    }  
}
```

When referring to a nested **enum**, precede it with the name of the containing class

M2.1_enums nested CarTester

```
public class CarTester {  
    public static void main(String[] args) {  
        Car myDreamCar = new Car("Ferrari",  
            Color.RED,  
            Car.Paint.METALLIC);  
    }  
}
```

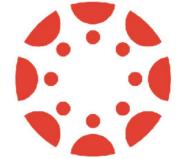
Assignment

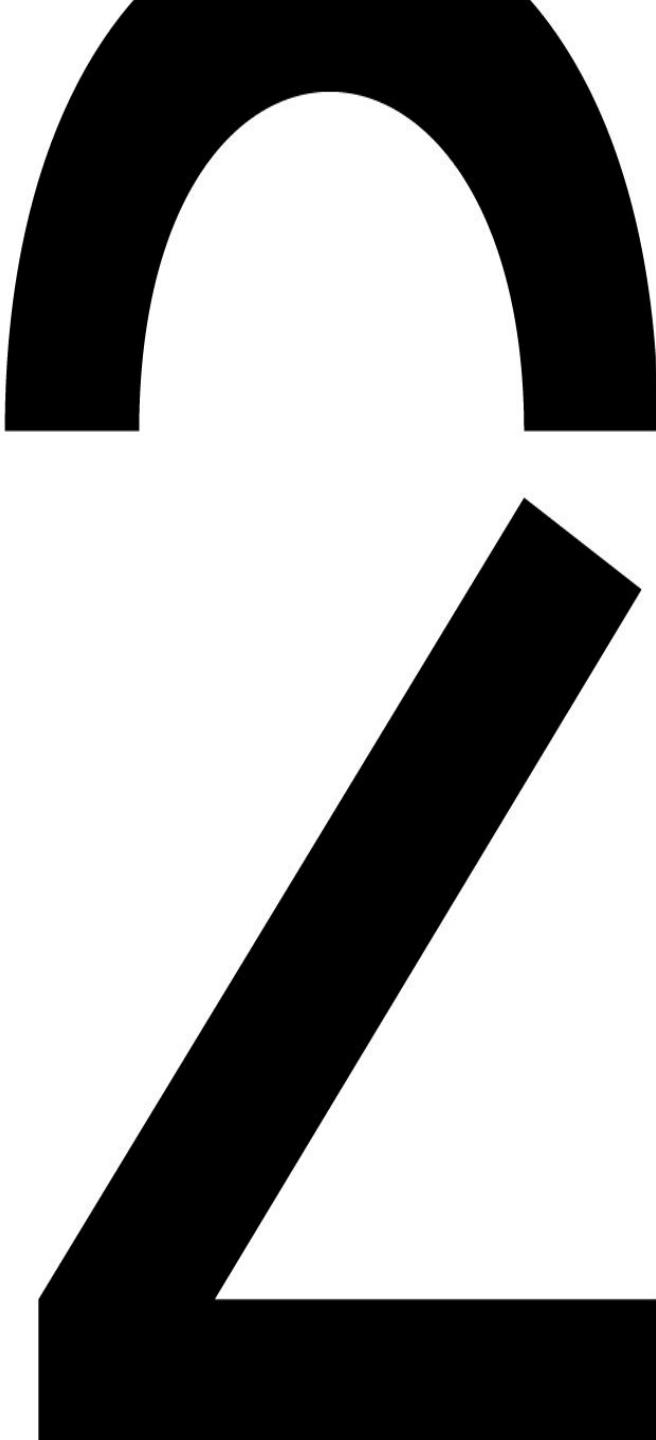


Exercise

Complete these assignments:

- Java 2.01
- Java 2.02
- Java 2.03

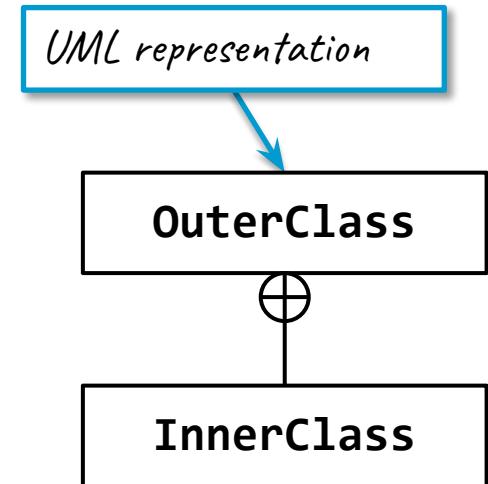




Inner classes

Inner classes

- In Java, it's possible to declare a class within another class. This is called an **inner class** or nested class.
- There are three kinds:
 - “regular” inner classes
 - static inner classes
 - anonymous inner classes
 - (not part of this module)
- Useful when creating objects that are linked to just *one* other object (regular/anonymous inner class) or class (static inner class).



Inner classes

M2.2_inner_classes bank.BankAccount

```
public class BankAccount {  
    public class Safe {  
        private int number;  
  
        public Safe(int number) {  
            this.number = number;  
        }  
  
        public int getNumber() {  
            return number;  
        }  
  
        @Override  
        public String toString() {  
            return "Safe number " + number;  
        }  
  
    }  
  
    private String iban;  
    private double balance;  
    private Safe safe;
```

```
    public Safe getSafe() {  
        return safe;  
    }  
  
    public BankAccount(String iban, double balance,  
                       int number) {  
        this.iban = iban;  
        this.balance = balance;  
        safe = new Safe(number);  
    }  
  
    public int getNumber() {  
        return safe.number;  
    }  
  
    @Override  
    public String toString() {  
        return "IBAN: " + iban + " Balance: " +  
               balance + " Number: " + safe.number;  
    }  
}
```

From within Safe, you can access all of BankAccount's attributes directly, even private ones!

Direct access to the private attribute `number`, without using the public accessor.

Inner classes

```
M2.2_inner_classes bank.BankAccountDemo
```

```
public class BankAccountDemo {  
    public static void main(String[] args) {  
        BankAccount account1 = new BankAccount("BE17 0000 0000 2121",  
                                              1000, 142857);  
        System.out.println(account1);  
        System.out.println(account1.getNumber());  
        System.out.println(account1.getSafe());  
  
        BankAccount.Safe safe = account1.new Safe(4662);  
        System.out.println(safe);  
        System.out.println(safe.getNumber());  
    }  
}
```

```
IBAN: BE17 0000 0000 2121 Balance: 1000.0 Number: 142857  
142857  
Safe number 142857  
Safe number 4662  
4662
```

You can create multiple instances of an inner class. Each of these instances are, at all times, connected to a specific instance of the outer class.

Inner classes

- Creating an instance of an inner class
 - From *within* the outer class:

```
public class OuterClass {  
    // ...  
    class InnerClass {  
        // ...  
    }  
    // ...  
    InnerClass inner = new InnerClass();  
}
```

- From *outside of* the outer class:

```
OuterClass outer = new OuterClass();  
OuterClass.InnerClass innerOut = outer.new InnerClass();
```

outer.new: Note that it's only possible to create instances of an inner class starting from an *instance* of the outer class.

Inner classes

- Using **this** within an inner class:
 - **this** refers to the object of the nested class
 - If you want to refer to the related object of the outer class, use **OuterClass.this**
- Possible access modifiers of inner classes:

Since an inner class is considered a *member* of the outer class, the following are possible:

 - **final**
 - **abstract**
 - **public** | **(package)** | **protected** | **private**
 - **static**

Assignment



Exercise

Complete these assignments:

- Java 2.04
- Java 2.05



Static nested classes

- Nested classes can be declared as **static**:

```
public class OuterClass {  
    public static class InnerClass {  
        // ...  
    }  
}
```

- Just like **static** variables or methods, they belong to the containing class (outer class).
- You can create an instance of a **static** nested class from the outer **class** (no outer class instance needed):

```
OuterClass.InnerClass nested = new OuterClass.InnerClass();
```

- Nested enums are implicitly **static**

Static nested classes

- The Map interface contains a **static nested interface** called 'Entry':

```
M2.2_inner_classes map_entry.MapEntryDemo
```

```
public class MapEntryDemo {  
    public static void main(String[] args) {  
        Map<String, String> messages = Map.ofEntries(  
            Map.entry("en", "Hello!"),  
            Map.entry("es", "¡Hola!"),  
            Map.entry("de", "Hallo!")  
        );  
  
        for (Map.Entry<String, String> entry : messages.entrySet()) {  
            System.out.printf("%s --> %s%n",  
                entry.getKey(),  
                entry.getValue());  
        }  
    }  
}
```

```
de --> Hallo!  
en --> Hello!  
es --> ¡Hola!
```



Exceptions

Catching exceptions

- Exceptions are said to be **thrown** ...
 - ... either by the JVM
 - NullPointerException
 - ArrayIndexOutOfBoundsException
 - ... by an API
 - InputMismatchException
 - ... or by ourselves
 - See “Programming 2”

Programming 1





Catching exceptions

- The try - catch statement

```
try {  
    System.out.print("Please enter an integer: ");  
    int number = scanner.nextInt();  
} catch (InputMismatchException exc) {  
    System.out.println("That's not a valid integer!");  
}
```

- The exception is **caught** in the catch-handler
 - Program won't crash
 - Program is able to properly inform the user of what happened

Multiple exceptions - chaining catch blocks

M2.3_exceptions catching.CatchChainDemo

```
public class CatchChainDemo {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter 2 numbers for which you want to  
calculate the remainder of the division: ");  
        try {  
            int number1 = scanner.nextInt();  
            int number2 = scanner.nextInt();  
            int remainder = number1 % number2;  
            System.out.printf("%d %% %d = %d",  
                number1, number2, remainder);  
        } catch (InputMismatchException ex) {  
            System.out.println("All numbers must be integers!");  
        } catch (ArithmaticException ex) {  
            System.out.println("Cannot divide by zero");  
        }  
    }  
}
```

1) Possible InputMismatchException
if input is not an integer

2) Possible ArithmaticException if
the second number is a zero

3) 2 catch blocks, each with their
own exception and logic

Multiple exceptions - multicatch

M2.3_exceptions catching.MultiCatchDemo

```
public class MultiCatchDemo {  
    public static void main(String[] args) {  
        Scanner scanner = new Scanner(System.in);  
        System.out.print("Enter 2 numbers for which you want to  
calculate the remainder of the division: ");  
        try {  
            int number1 = scanner.nextInt();  
            int number2 = scanner.nextInt();  
            int remainder = number1 % number2;  
            System.out.printf("%d %% %d = %d",  
                number1, number2, remainder);  
        } catch (InputMismatchException | ArithmeticException ex) {  
            System.out.println("Something went wrong");  
        }  
    }  
}
```

1) Possible InputMismatchException if input is not an integer

2) Possible ArithmeticException if the second number is a zero

3) 1 catch block, catching 2 different exceptions that are both handled with the same logic

Throwing exceptions

- For now we've been catching exceptions thrown by the JVM and the Java API (e.g. `Scanner`).
- You can also throw exceptions yourself!
- Usage:

```
Exception exceptionObject = new Exception();
throw exceptionObject;

// ... shorter:
throw new Exception();

// ... with a message:
throw new Exception("Error message");
```

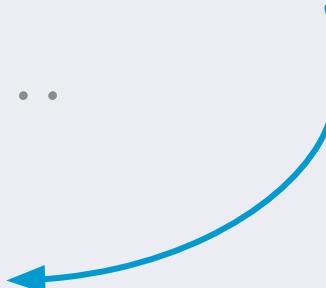
In this example we're creating an object of the class `Exception`. It's recommended to use a more specific exception subclass.

Throwing exceptions

- You can catch the exception within the method using a try-catch statement...

```
public void method() {  
    try {  
        if (!condition) {  
            throw new Exception("Something went wrong...");  
        } else {  
            // Execute statements...  
        }  
    } catch (Exception ex) {  
        // Handle exception...  
    }  
}
```

A message can be passed to the Exception constructor.



Throwing exceptions

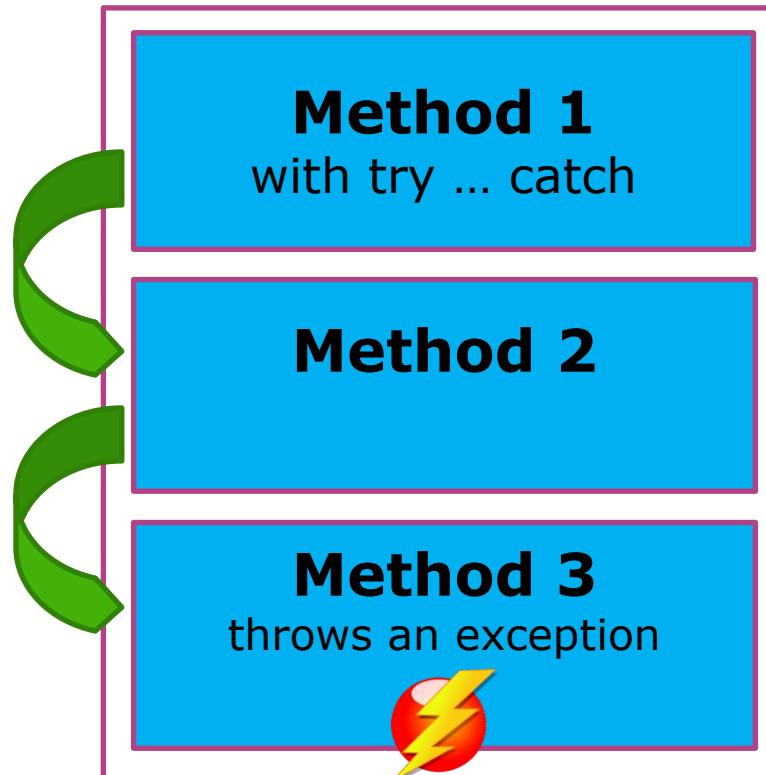
- ... or the method can **throw the exception** to the method that called this one.

Add to your method declaration.

```
public void myMethod() throws Exception {  
    if (!condition) {  
        throw new Exception("Something went wrong...");  
    } else {  
        // Execute statements...  
    }  
}
```

Where is an exception handled?

Call stack: Method 1 → Method 2 → Method 3

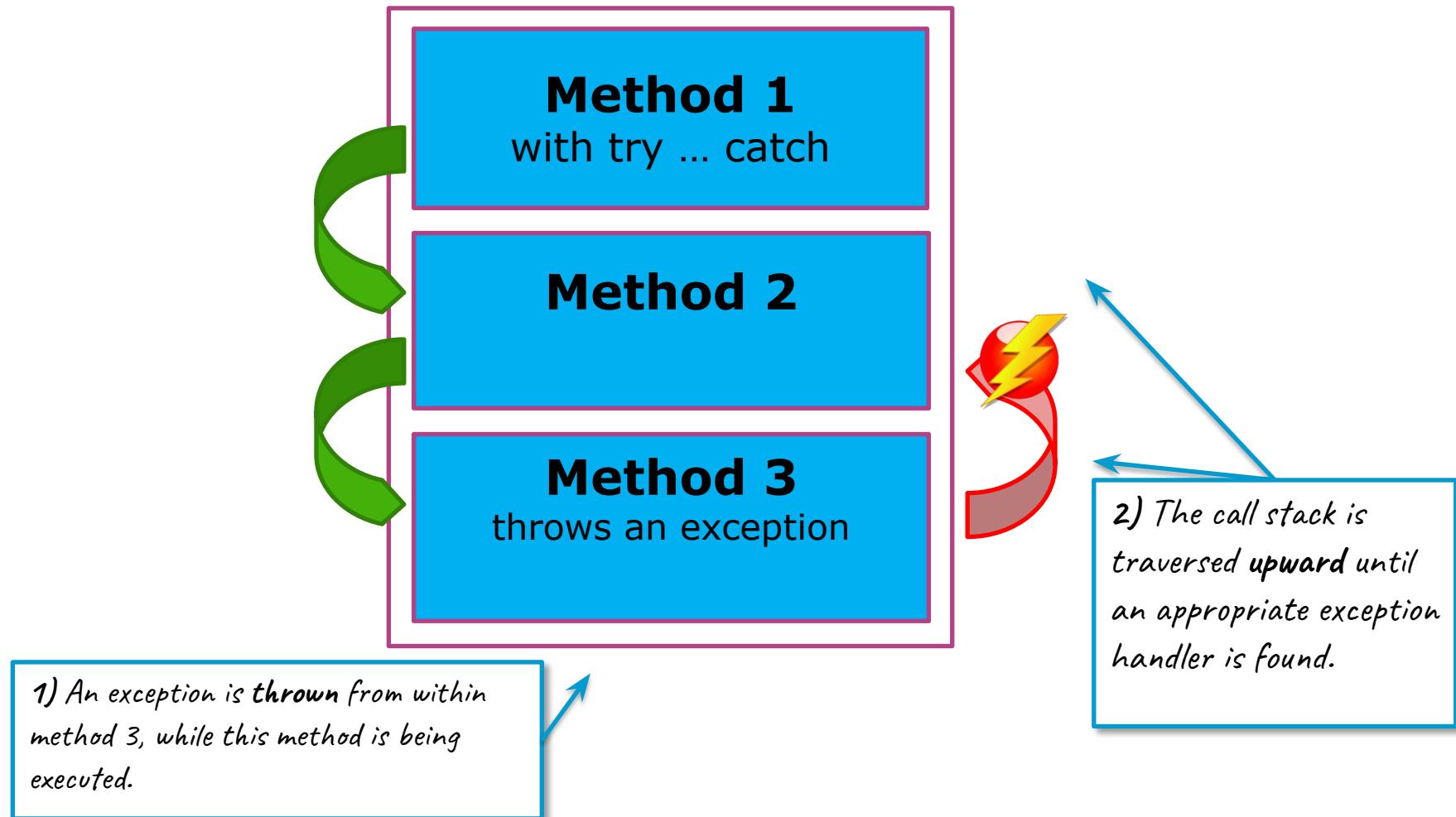


- 1) An exception is thrown from within method 3, while this method is being executed.



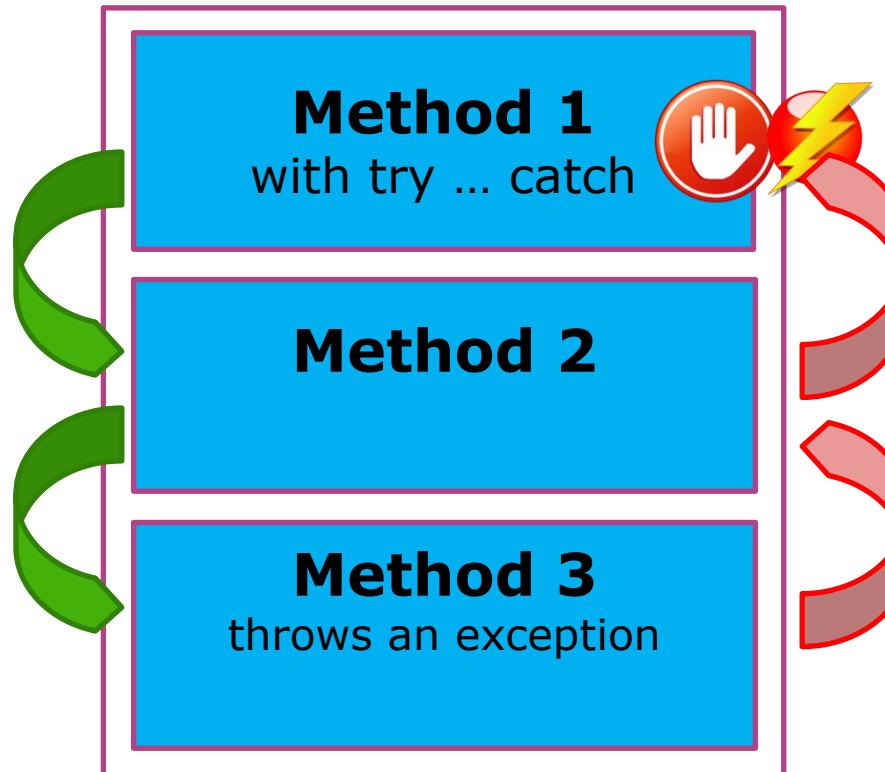
Where is an exception handled?

Call stack: Method 1 → Method 2 → Method 3



Where is an exception handled?

Call stack: Method 1 → Method 2 → Method 3



1) An exception is thrown from within method 3, while this method is being executed.

4) In case no exception handler was found in the call stack, the application is terminated (crash).

3) The method with the exception handler catches the exception and takes appropriate action.

2) The call stack is traversed upward until an appropriate exception handler is found.

Call stack: example

```
M2.3 exceptions callstack.CallstackDemo
```

```
public class CallstackDemo {  
    public static void main(String[] args) {  
  
        Book lotr = new Book("TT", "A 20th century classic.",  
                             19.90, "J.R.R. Tolkien",  
                             "The Two Towers");  
  
        public Book(String code, String description, double price,  
                   String author, String title) {  
  
            protected Product(String code, String description, double price) {  
                if (code == null || code.length() < 3) {  
                    throw new IllegalArgumentException("Code must be at least 3 chars.");  
                }  
                this.code = code;  
                this.description = description;  
                this.price = price;  
            }  
  
            this.author = author;  
            this.title = title;  
        }  
    }  
}
```

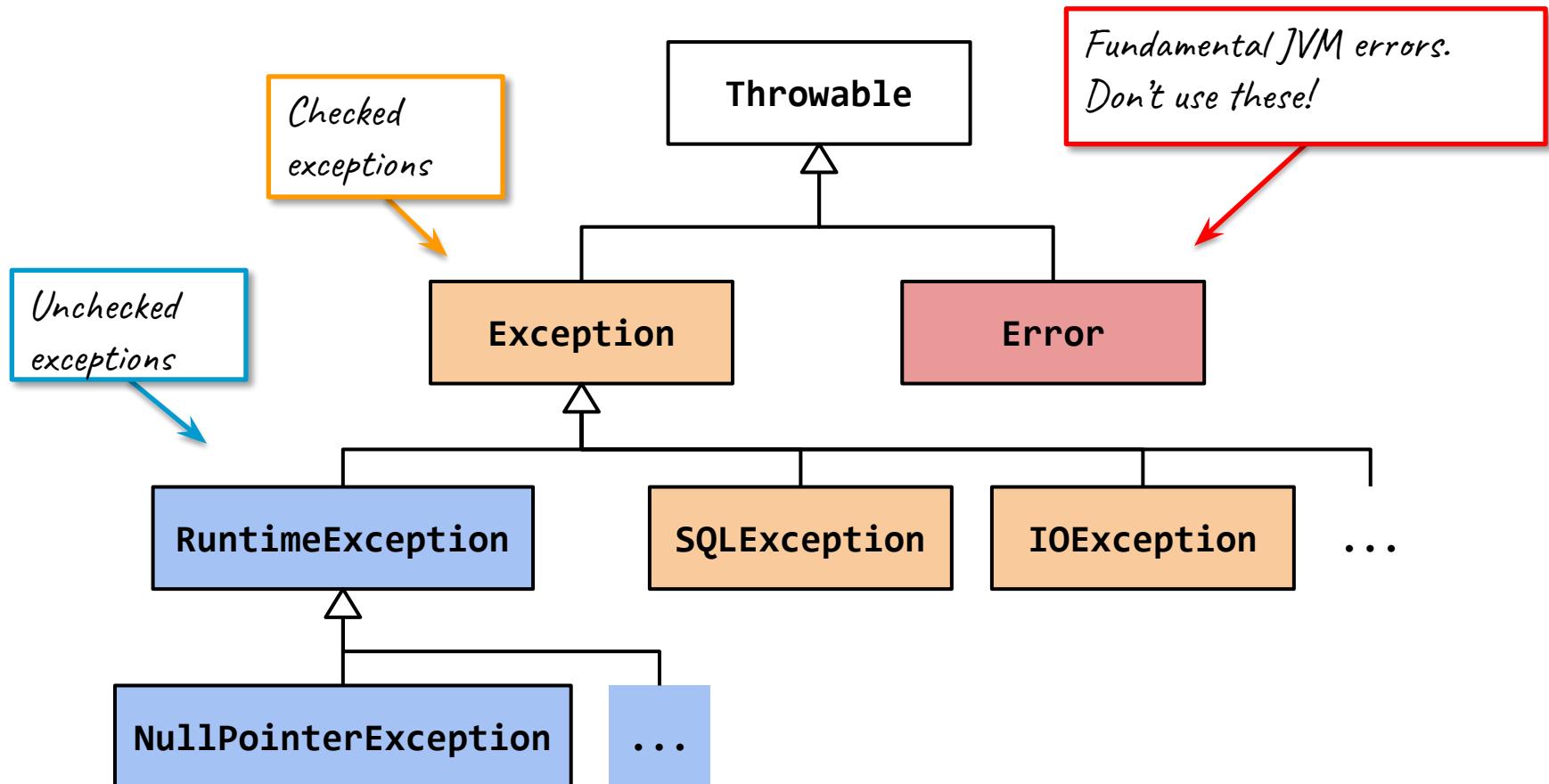
Exception in thread "main" java.lang.IllegalArgumentException: Code must be at least 3 characters

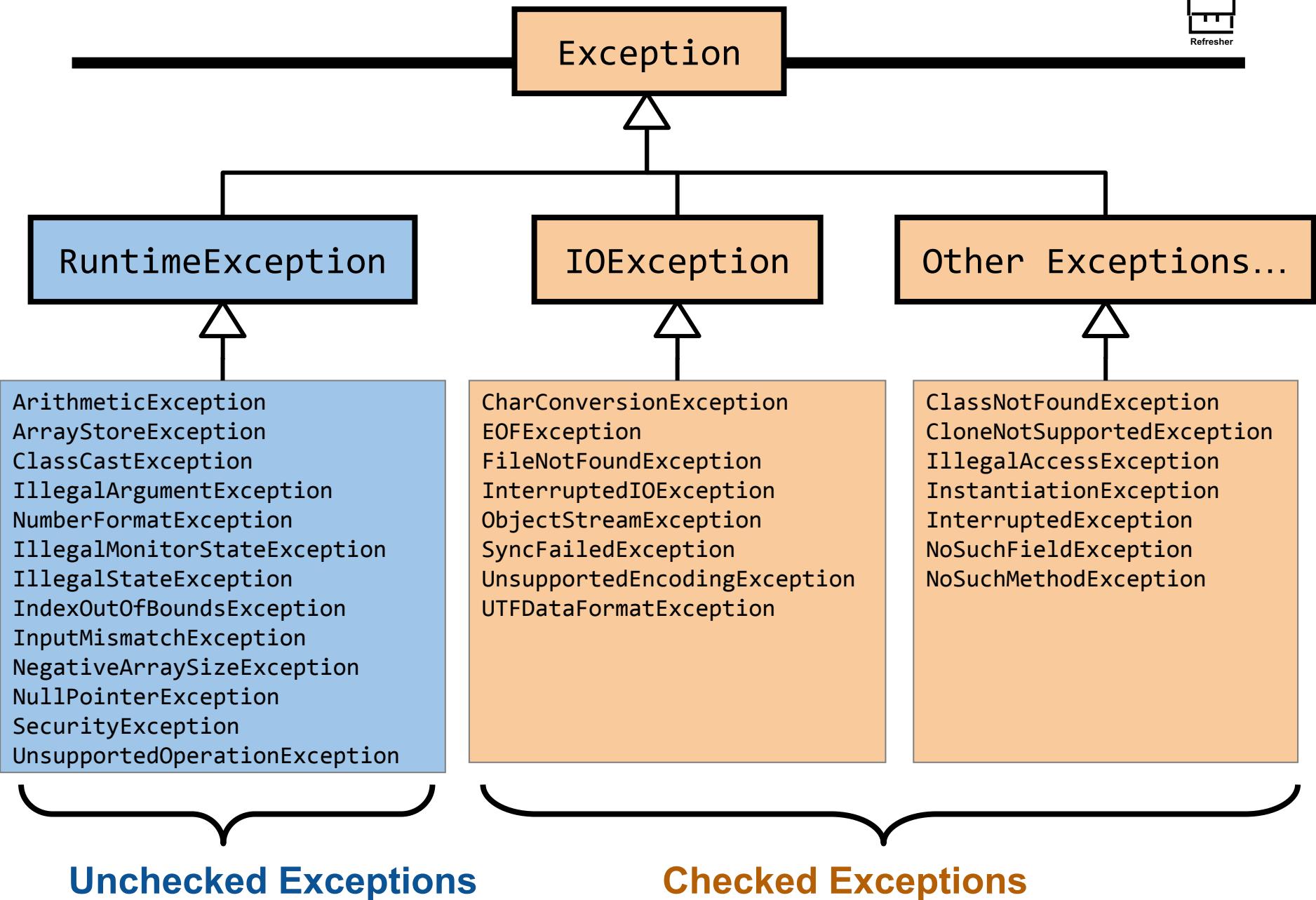
at be.kdg.pro12.Product.<init>(Product.java:18)

at be.kdg.pro12.Book.<init>(Book.java:11)

at be.kdg.pro12.DemoCallStack.main(DemoCallStack.java:10)

Types of exceptions





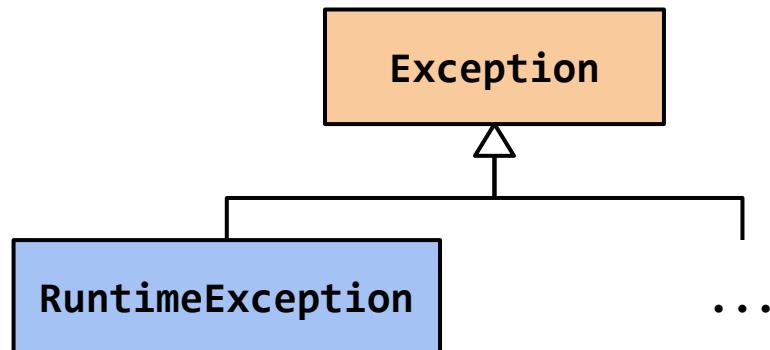


Types of exceptions

- **Unchecked** exceptions
 - Catching these is **optional**
 - Catching these is *not* checked by the compiler
 - Many of these from the Java API shouldn't ever happen (→ programming errors)
 - Examples: `IndexOutOfBoundsException`,
`NullPointerException`, `ArithmaticException`, ...
- **Checked** exceptions
 - Catching these is **mandatory**: either the method itself or further up the call stack
 - Catching these *is* checked by the compiler
 - Must be mentioned in the method declaration if a method throws such an exception (**throws**)
 - Examples: `FileNotFoundException`, `SQLException`,
`InterruptedException`, ...

Custom exceptions

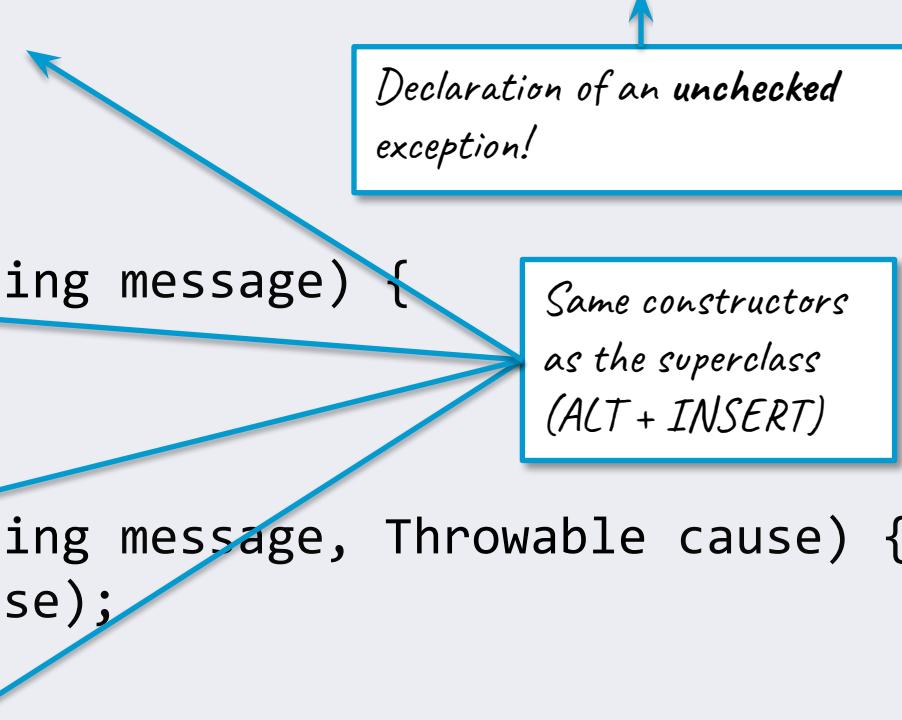
- It's possible to write **custom** exception classes.
 - Only when existing exception classes aren't appropriate!
- A custom exception class should extend from an existing exception class.
 - Extend **Exception** to create a **checked** exception.
 - Extend **RuntimeException** to create an **unchecked** exception.
 - Use these if you do not want to force the caller to catch the exception. Most commonly used.



Custom exception classes

- Generally, exceptions have four constructors:

```
public class MyException extends RuntimeException {  
    public MyException() {  
        super();  
    }  
  
    public MyException(String message) {  
        super(message);  
    }  
  
    public MyException(String message, Throwable cause) {  
        super(message, cause);  
    }  
  
    public MyException(Throwable cause) {  
        super(cause);  
    }  
}
```



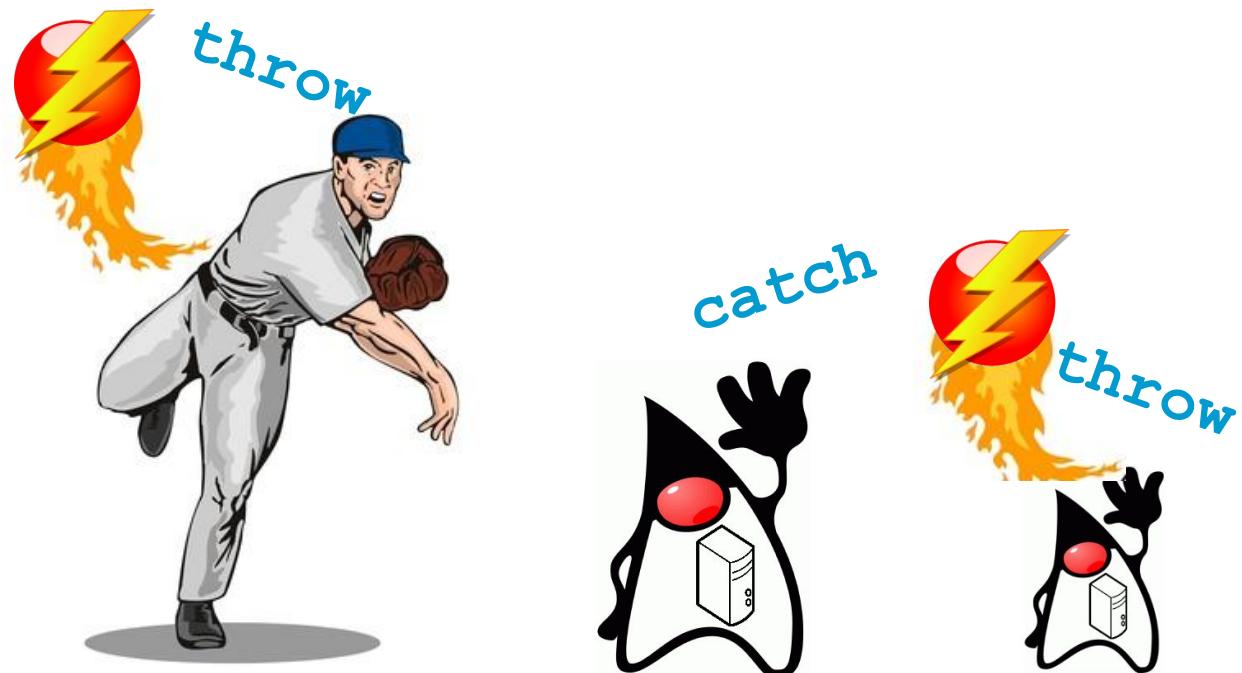
Declaration of an unchecked exception!

Same constructors as the superclass (ALT + INSERT)

Catching, wrapping, and rethrowing

- If we don't want to handle an exception ourselves, we can still choose to use our custom exception class:

1. Catch
2. Wrap in custom Exception
3. Rethrow



Catching and rethrowing

```
public void method() throws Exception {  
    try {  
        // Code that may throw an exception  
    }  
    catch(Exception ex){  
        System.out.println(ex.getMessage());  
        throw ex;  
    }  
}
```

1) Catch the exception

2) Rethrow the exception

(In this example, no wrapping yet.)

When to use:

- When you want to add some information to an exception.
- When you want to do *some* error handling, but still want to allow further error handling.

Catching, wrapping, and rethrowing

```
public void method() throws MyException {  
    try {  
        // Code that may throw an exception  
    }  
    catch(Exception ex) {  
        // Some error handling specific to 'method'  
        throw new MyException(ex);  
    }  
}
```

1) Catch the exception

```
public void doSomething() {  
    try {  
        method();  
    }  
    catch (MyException ex) {  
        System.out.println("MyException: Problem!");  
        System.out.println(ex.getCause().toString());  
    }  
}
```

2) The original exception (root cause) is wrapped in a custom exception and rethrown

3) The caller catches MyException

4) The root cause can be retrieved using the `getCause` method

Assignment



Exercise

Complete these assignments:

- Java 2.06
- Java 2.07
- Java 2.08
- Java 2.09





JavaFX setup

Installation



- Download JavaFX from:
<https://gluonhq.com/products/javafx/>
- Download the **SDK** for your platform (LTS release 21)

Downloads

JavaFX version: 21.0.2 [LTS] Operating System: [any] Architecture: [any] Type: **SDK**

Include older versions

Supported Platforms

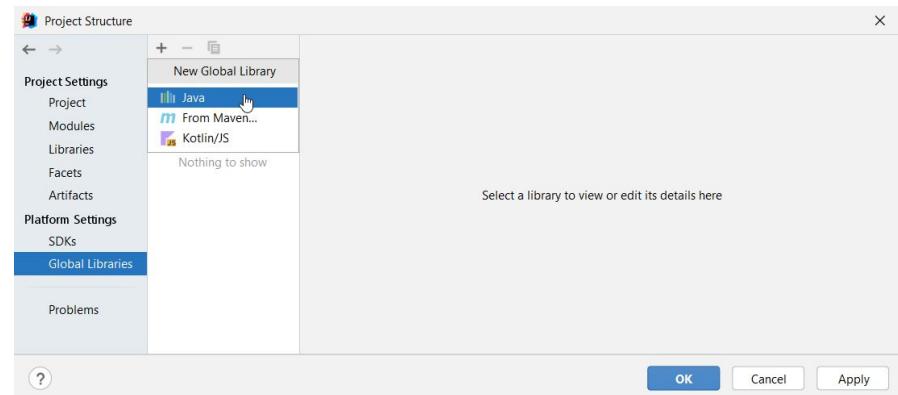
OS	Version	Architecture	Type	Download
Linux	21.0.2	x64	SDK	Download [SHA256]
macOS	21.0.2	aarch64	SDK	Download [SHA256]
macOS	21.0.2	x64	SDK	Download [SHA256]
Windows	21.0.2	x64	SDK	Download [SHA256]

Javadoc can be downloaded here as well

- Unzip to a local directory
 - Recommended: lib/javafx in your home directory
(example C:\Users\myusername\lib\javafx)
 - Don't put it in your downloads folder.

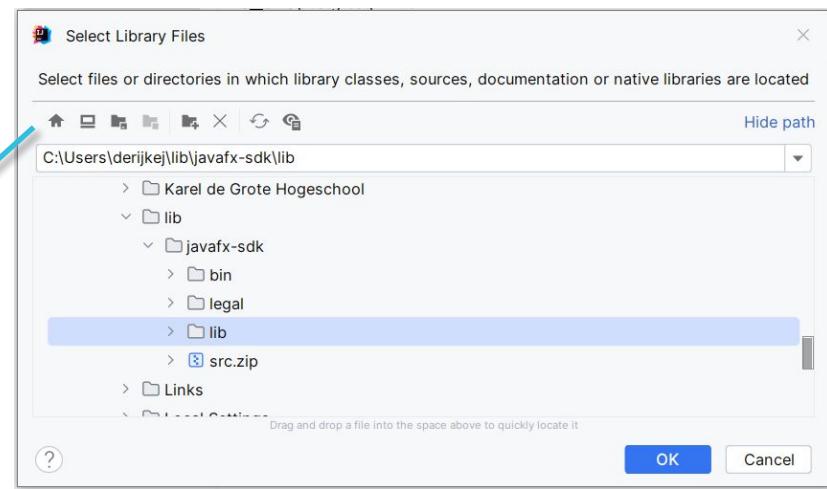
One time setup

1. Create a new Java project
 - uncheck "add sample code"
2. In the project, go to *File* → *Project Structure* → *Global Libraries* → + → *Java*



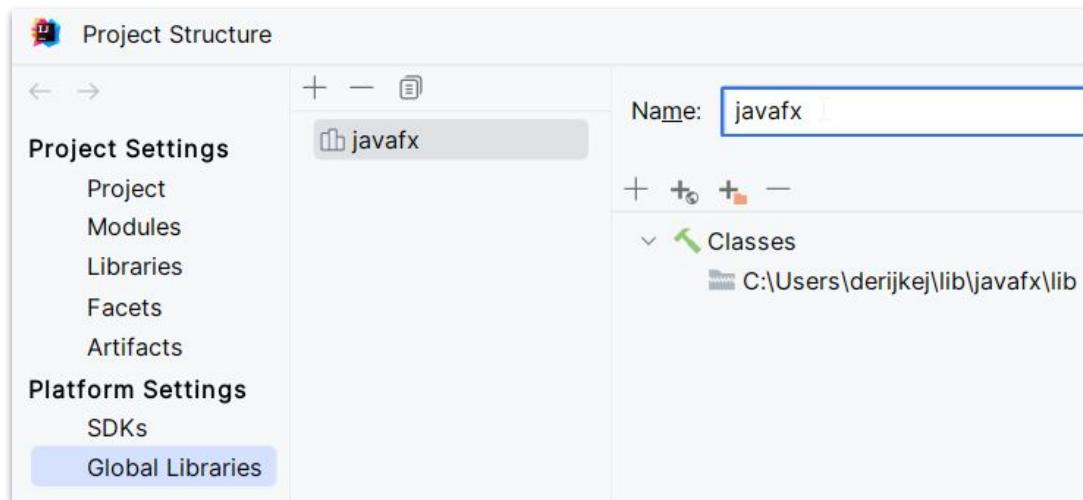
3. Select the `lib` directory under your JavaFX installation

Takes you to your home directory



One time setup

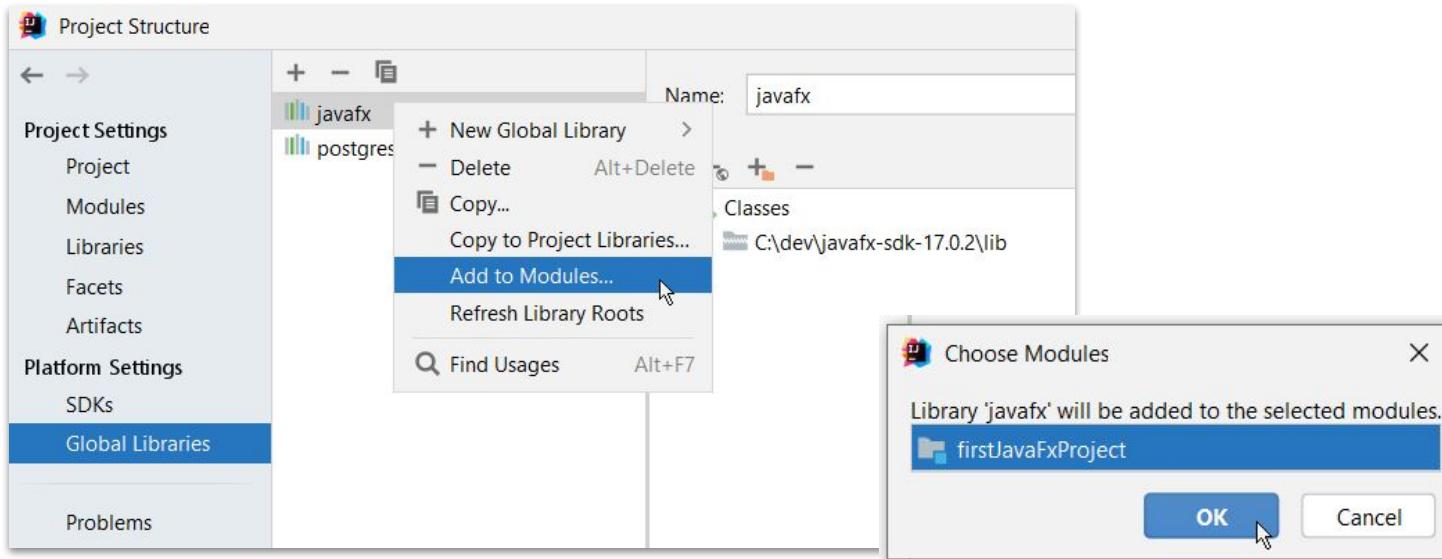
4. IntelliJ will ask you which module the library should be added to. **Click cancel.**
5. Back on the Project Structure screen, change the library name from lib to javafx:



Project Setup (to do in each JavaFx project)

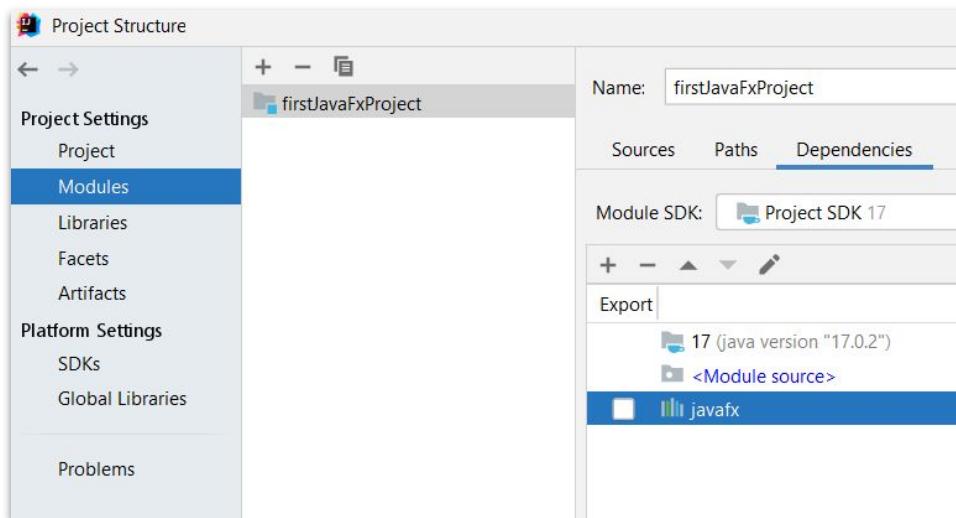
Starting from here, the steps have to be done in every javafx project

1. Right click the javafx global library and click "Add to modules..."
2. Pick the module with your project name and click OK



Project Setup

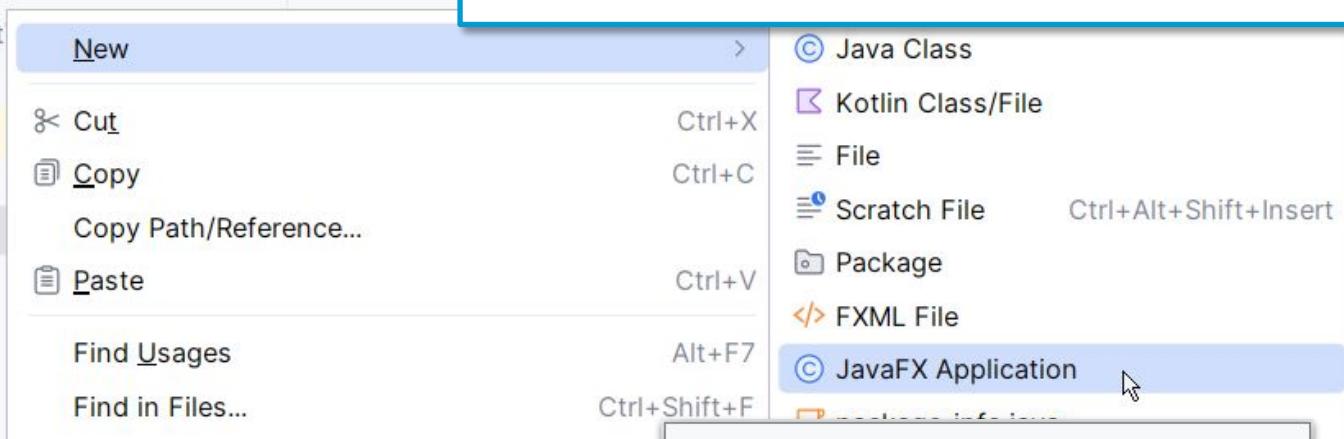
- Optionally you can verify in modules | project name | dependencies that the JavaFx library is present
 - alternatively, you can add the library to the module from here



Hello World

Project ▾

```
firstJavaFxProject C:\project
  > .idea
  > out
  > src
    > be.kdg.pro2.javafx
      firstJavaFxProject.iml
  > External Libraries
  > Scratches and Consoles
```



```
package be.kdg.pro2.javafx;

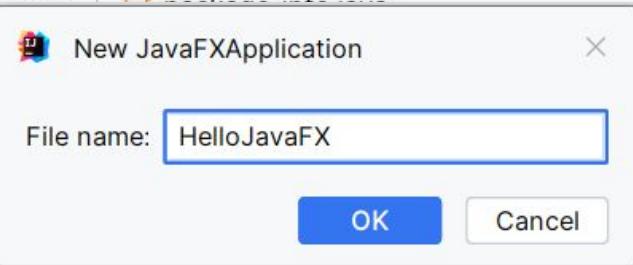
import javafx.application.Application;
import javafx.stage.Stage;

public class HelloJavaFX extends Application {

    public static void main(String[] args) {Launch(args); }

    public void start(Stage primaryStage) {}

}
```



Hello World

```
package be.kdg.pro2.javafx;

import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.control.Label;
import javafx.scene.layout.BorderPane;
import javafx.stage.Stage;

public class HelloJavaFX extends Application {
    @Override
    public void start(Stage primaryStage) {
        Label helloLabel = new Label("Hello JavaFX!");
        BorderPane root = new BorderPane(helloLabel);
        Scene scene = new Scene(root);
        primaryStage.setScene(scene);
        primaryStage.show();
    }

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

Add the highlighted code below to the start method

You'll need to import some classes.

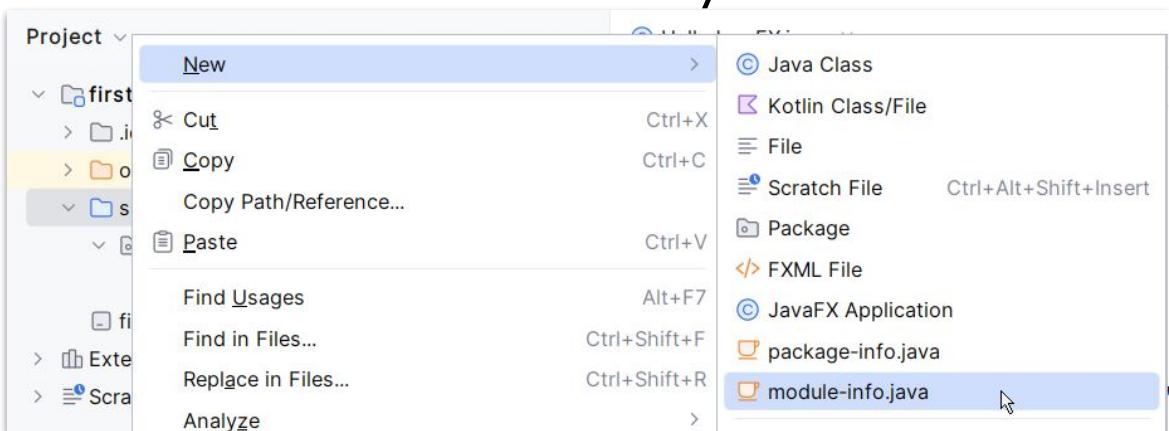
Take care you import classes with javafx top package!

Launch the application

Execute the main method. An error message is shown:

Error: JavaFX runtime components are missing, and are required to run this application

You'll have to tell your application to use the JavaFX modules.
Add a **module-info.java** file to the src directory



The screenshot shows a right-click context menu in an IDE. The 'New' option is selected. A list of file types is shown, with 'module-info.java' highlighted at the bottom.

module-info.java

```
open module first javafx project {  
    requires javafx.controls;  
    requires javafx.media;  
    requires javafx.graphics;  
    requires javafx.base;  
}
```

Add module-info.java
to the src-folder
Add at least the
highlighted content

Launch the application

Run the application again



Congratulations!

You are ready to build Java Desktop applications now!

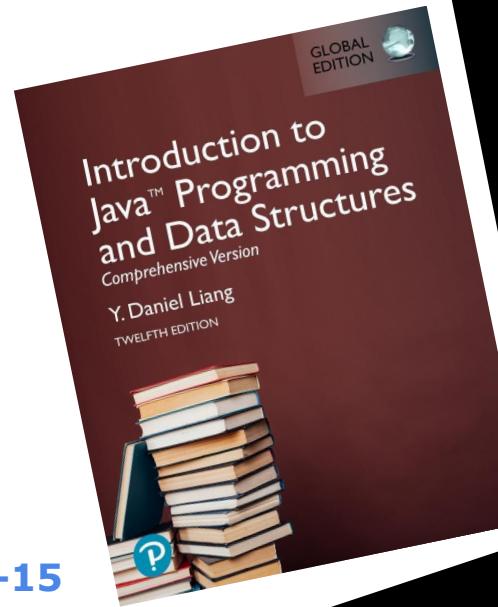


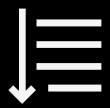
- enums
- Inner classes
 - Nested classes
 - Static nested classes
- Exceptions
 - Types of exceptions
 - Throwing Exceptions
 - Custom exceptions
- JavaFX setup



Basics

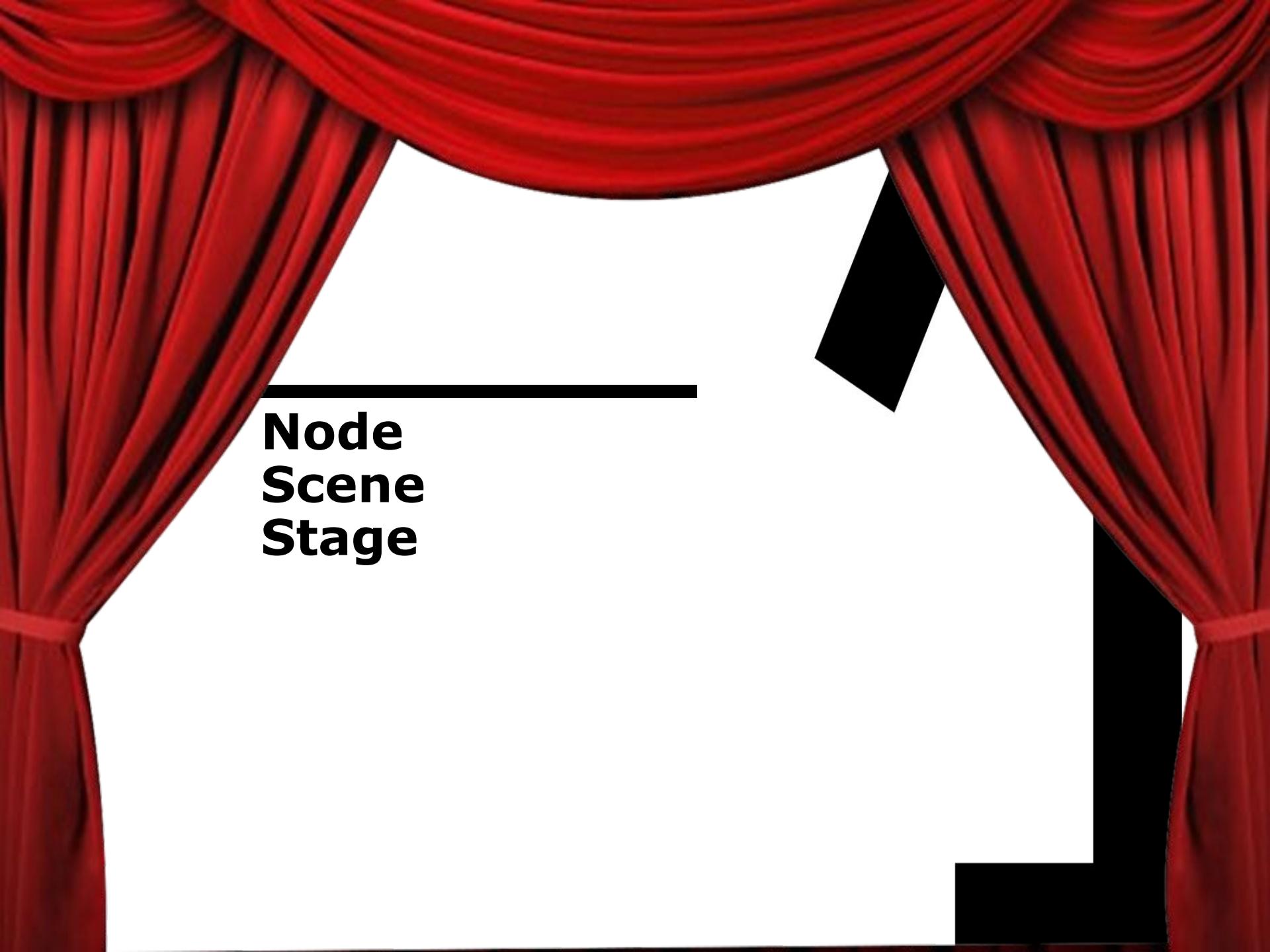
Programming 2





Agenda

1. Node / Scene / Stage
2. A GUI application cookbook:
Screen Reader
3. Lambda expressions
4. The model
5. Model View Presenter
6. Make an MVP template project

A red curtain is drawn back to reveal a white stage area. There are three black rectangular blocks on the stage: one horizontal bar near the top left, one triangular block near the center right, and one L-shaped block at the bottom right.

**Node
Scene
Stage**

What is JavaFX?

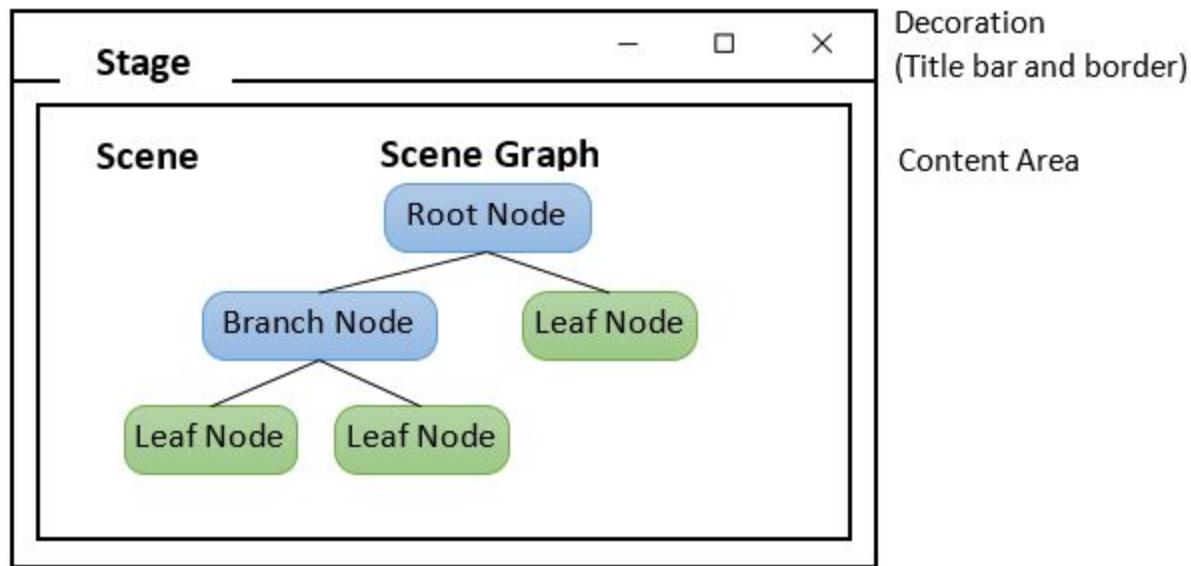
- Java Library (API) to build desktop applications with a Graphical User Interface (GUI)
 - Can be used for mobile applications (Android, iOS...)
-  <https://openjfx.io/>
 - Not included in JDK (since JDK 11)
 - Alternative for JDK Swing/AWT libraries
- Based on modern technologies:
 - CSS, XML, Scenegraphs, Transforms, Animations, 3D, ...

Theatre metaphor

1. Nodes are basic GUI elements
2. Compose a scene by organising nodes in a tree structure (the scenegraph)
3. `show()` the scene on the stage (in a window)



Theatre Metaphor



Hello World

```
M1HelloWorld/HelloWorld
```

```
import javafx.application.Application;  
import javafx.scene.*;  
import javafx.stage.Stage;
```

extends abstract class Application. The abstract **start** method must be overridden.

```
public class HelloWorld extends Application  
    @Override
```

node

```
        public void start(Stage primaryStage) {
```

root node

```
            Label lblHello = new Label("Hello World!");  
            BorderPane root = new BorderPane(lblHello);
```

Scene

```
            Scene scene = new Scene(root);
```

Stage

```
            primaryStage.setScene(scene);
```

```
            primaryStage.show();
```

start receives a Stage (window) to run in

Set scenegraph root on Scene

A Stage can contain one Scene at a time

```
        }
```

```
        launch(args);
```

```
}
```

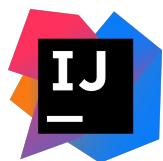
```
}
```

JavaFX import clashes

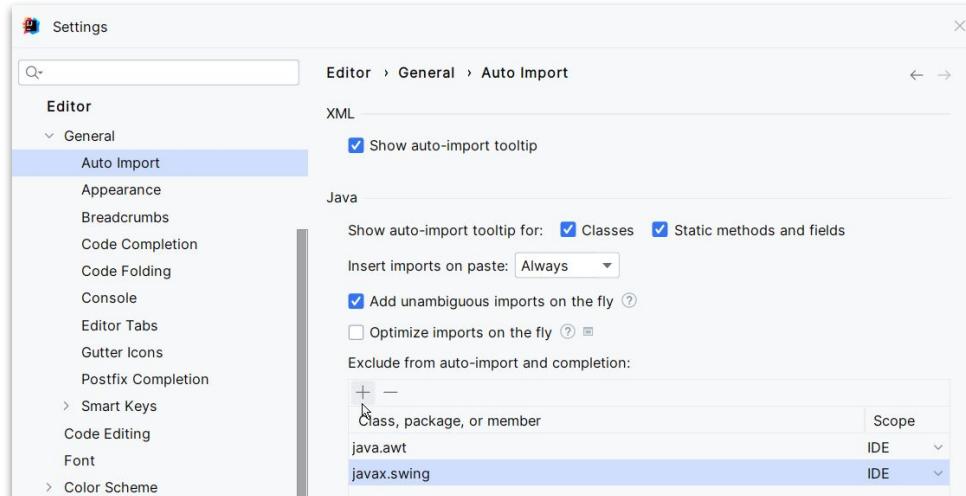


Heads up

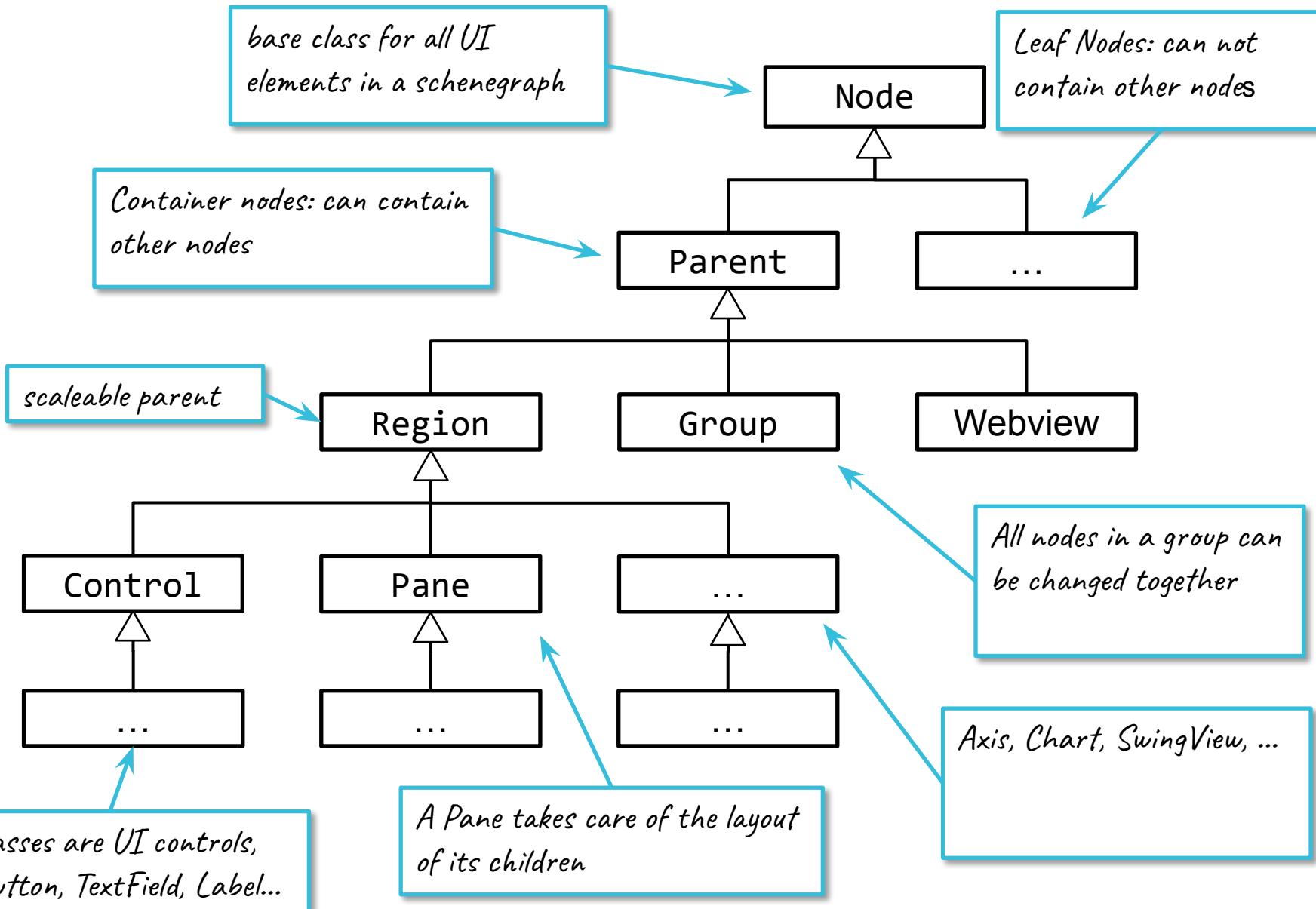
Mind importing from `javafx` packages and **NOT** from `java.awt` or `javax.swing`



- these are older GUI library in the JDK
- We are not going to use those in this course.
you can exclude them from IntelliJ autoimports using
 - File | Settings...
 - Editor | General | autoimport
 - Exclude from autoimport and completion
 - `java.awt`
 - `javax.swing`

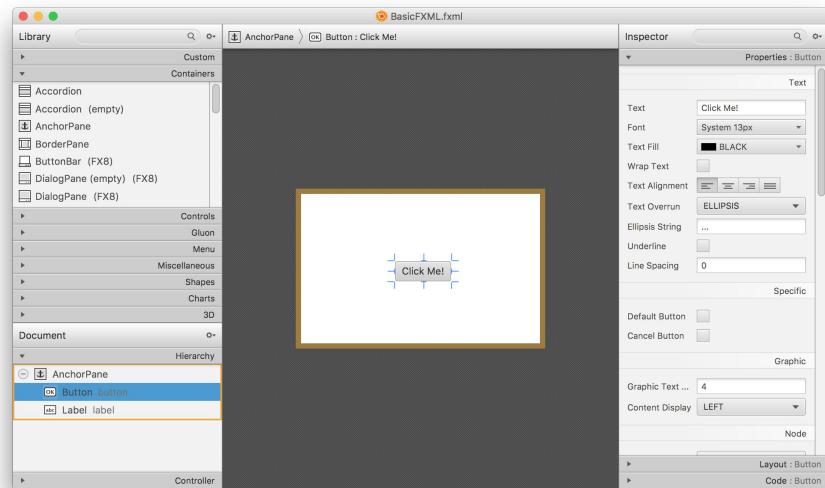


Node Inheritance Hierarchy



JavaFX Scene Builder

- JavaFX Scene Builder: a Visual Layout tool for Rapid Application Development by dragging and dropping GUI components
 - <https://gluonhq.com/products/scene-builder/>
 - The GUI is saved to an XML file (FXML)
This approach is taken by some books
- We will **NOT** use the JavaFX Scene Builder **NOR** FXML
- We will exclusively rely on **Java programming** using the **JavaFX API**



Self Learning

- NOT everything you need to make a fancy JavaFX application is in these slides nor in the book “*Introduction to Java Programming and Data Structures*” by Y.D. Liang (Chap. 14-16)
- Learn and research yourself, using additional resources
 - Documentation tab at openfx.io, especially
 - [javadoc: https://openfx.io/javadoc/version/](https://openfx.io/javadoc/version/)
 -  Heads up Many methods and attributes of JavaFX controls are inherited from superclasses. Look in the javadoc of the superclasses as well!
 - [Jenkov JavaFX Tutorial](#)
 - Google

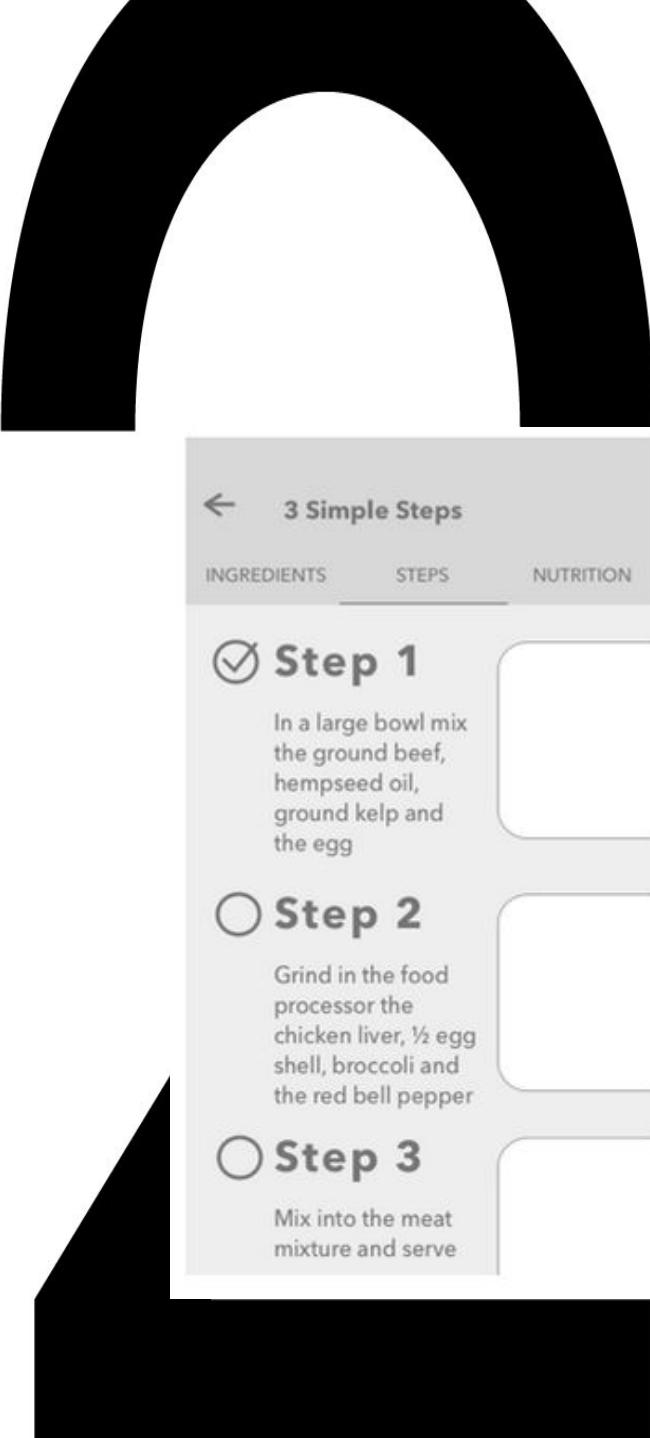
Exercises



Make exercise: Java 3.01

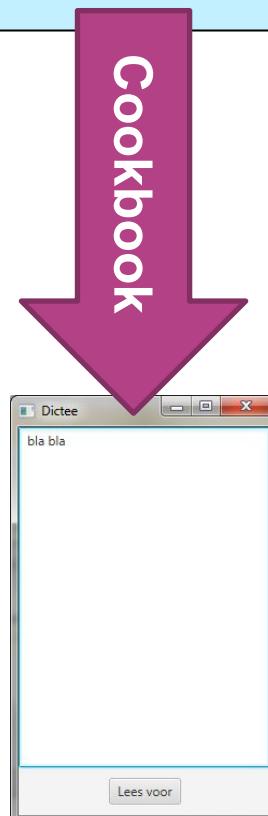


A GUI Application Cookbook: Building a Screen Reader



Your first GUI application: Screen Reader

*Write a simple UI in which the user can type a text.
When the user presses the button his text is read aloud.*



GUI application cookbook

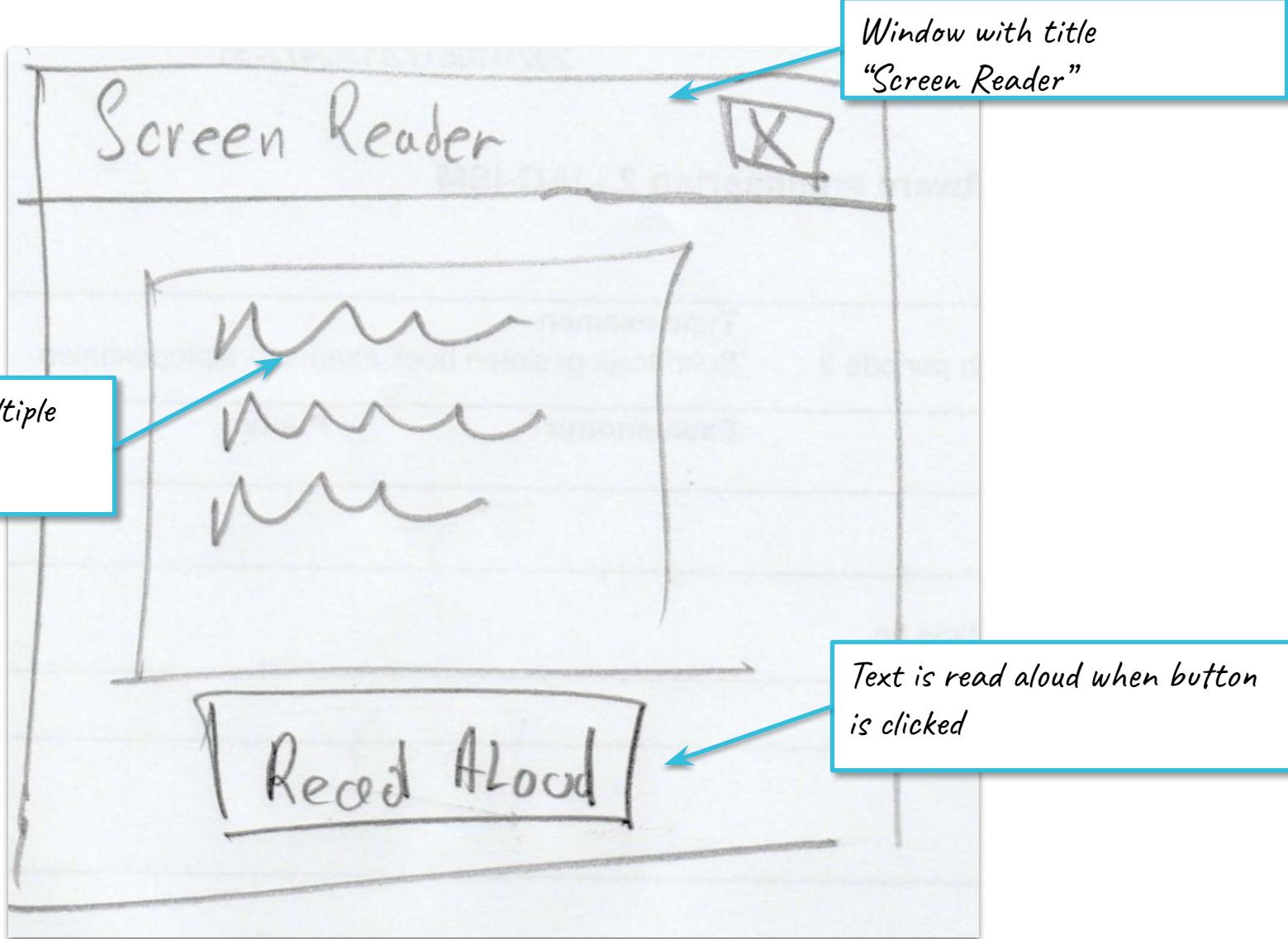
- 
1. Draw wireframe
 2. Build main screen
 3. Build UI
 - i. Initialise controls
 - ii. Layout using panes
 4. Handle Events
 - i. Write code to handle the event (do action, change GUI...)
 - ii. Attach code to event target (GUI control)

1: Draw wireframe

- Sketch the screen(s) of your application
 - which components?
 - layout
- Tools
 - pencil & paper
 - desktop: e.g. [Balsamic](#)
 - cloud: e.g. [Lucidchart](#)



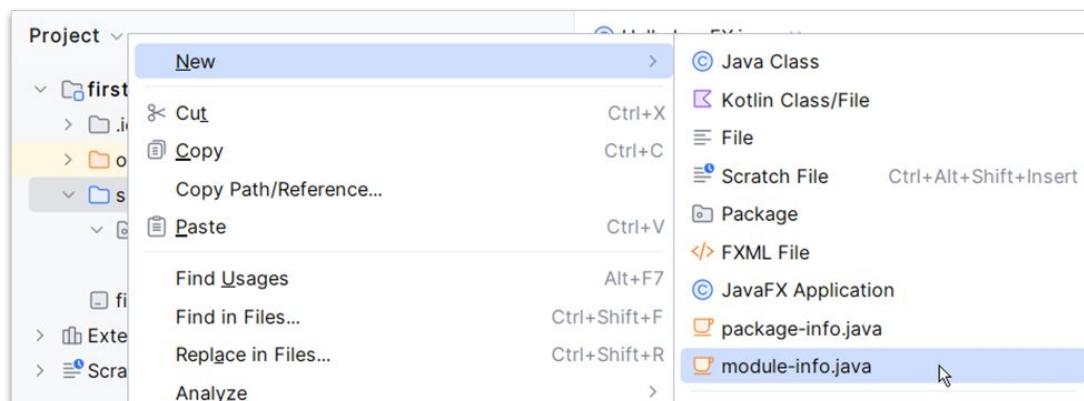
✓ 1: Draw wireframe



2. Build main screen



- Make a new IntelliJ project called **ScreenReader**
- Add the JavaFX libraries to the project
- Add the `module-info.java` file to the `src` directory



and adjust the content:

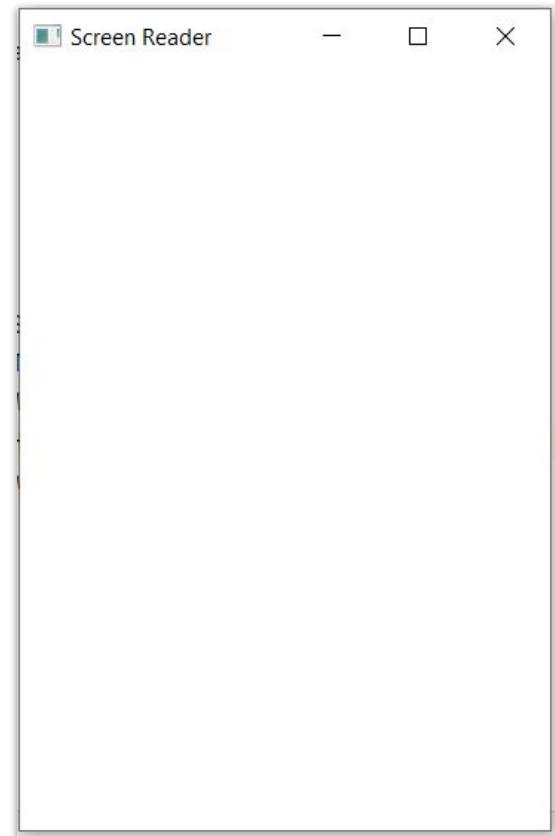
```
module-info.java
```

```
open module first javafx_project {  
    requires javafx.controls;  
    requires javafx.media;  
    requires javafx.graphics;  
    requires javafx.base;  
}
```

2. Build main screen



- Work in a package be.kdg.screenreader
- Size: 300x450
- Title as in wireframe
- Empty stage



✓ 2: Build main screen

M2ScreenReader/screenreader.Main

```
import javafx.application.Application;
import javafx.stage.Stage;

public class Main extends Application {

    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Screen Reader");
        primaryStage.setWidth(300);
        primaryStage.setHeight(450);
        primaryStage.show();
    }
}
```



3.1: initialise controls

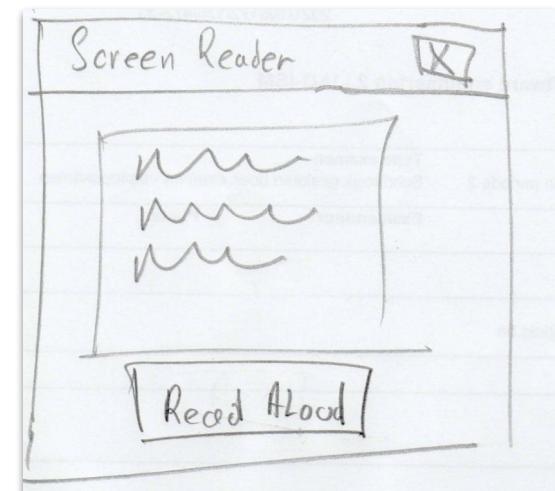


Exercise

- Which UI controls do you need?
 - TextField? (hint: starts with “Text”)
 - Button?
- All controls are in `javafx.scene.control`!

Plenty of controls! We'll discuss
the most important ones later.

We got 6 of those 😊
Check the doc



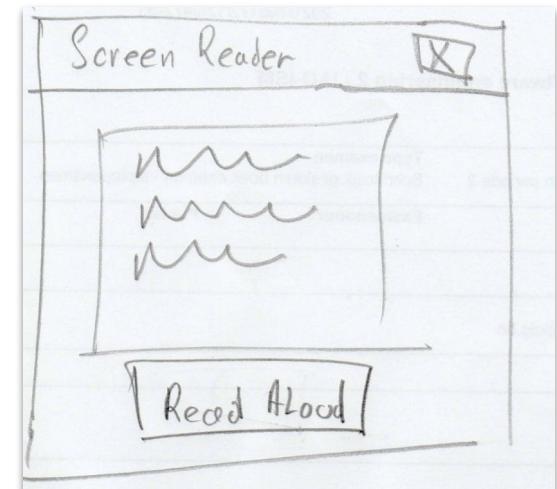
3.2: layout using panes



Exercise

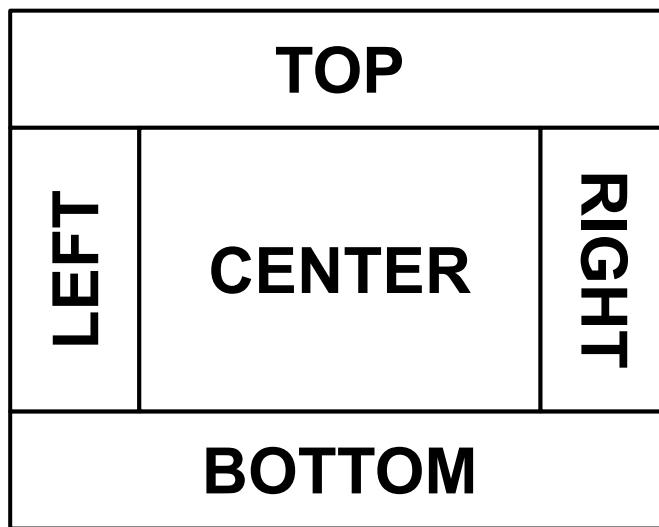
- Where to put controls?
 - Text: center, scale with window
 - Button: bottom-center, fixed size
- Position controls in a layout pane
 - We will use BorderPane
- All layout panes classes are in `javafx.scene.layout`!

We'll discuss layout panes in a later module.



3.2: BorderPane

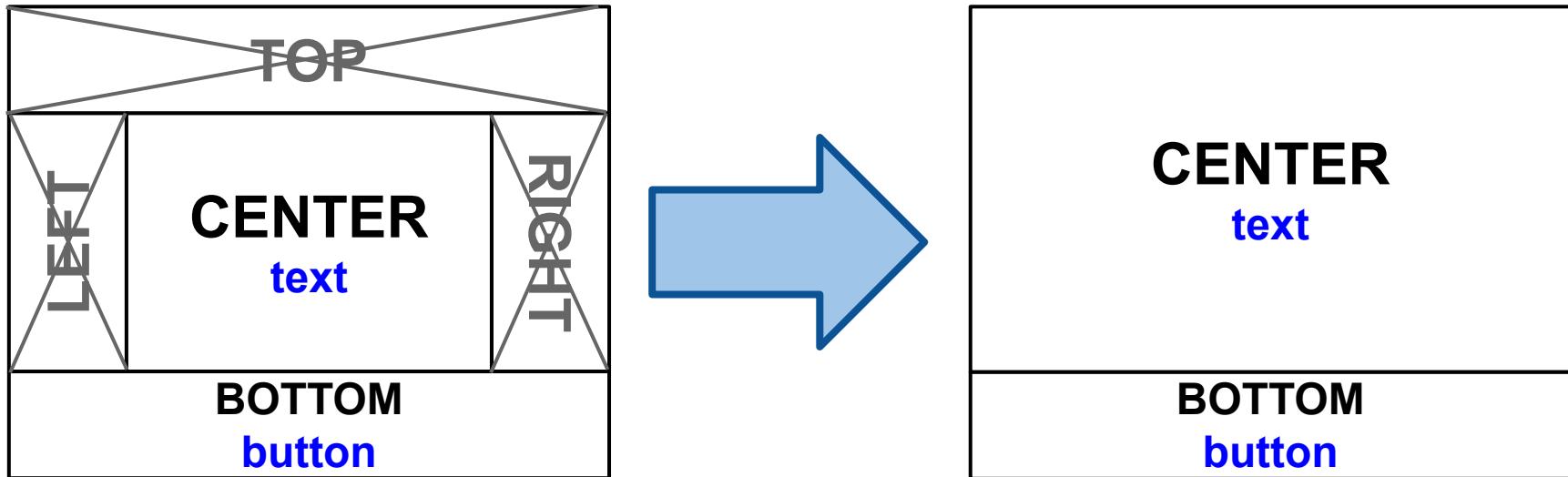
- BorderPane consists of **5 areas**: *top, right, bottom, left en center*.
- You can put **one node** in each area
 - the node can be a container



3.2: Write the code!



- Add the controls to the BorderPane
- Add the BorderPane to the Scene



3: Build UI

M2ScreenReader/screenreader.Main

```
public void start(Stage primaryStage) {  
    Button readButton = new Button("Read Aloud");  
    TextArea textArea = new TextArea("Enter text to read");  
    BorderPane root = new BorderPane();  
    root.setCenter(textArea);  
    root.setBottom(readButton);  
    BorderPane.setAlignment(readButton, Pos.CENTER);  
    BorderPane.setMargin(readButton,  
        new Insets(10, 10, 10, 10));  
    primaryStage.setTitle("Screen Reader");  
    primaryStage.setWidth(300);  
    primaryStage.setHeight(450);  
    primaryStage.setScene(new Scene(root));  
    primaryStage.show();  
}
```

Initialise controls

BorderPane
text in the center
button at the bottom

The start method does
everything?
Let's refactor!

Add scenegraph to Scene
Add Scene to Stage



3: Separate View class

Put scenegraph in separate View class
inherits from root element (BorderPane)
Put in view package

M2ScreenReader/screenreader.view.ScreenReaderView

```
package be.kdg.screenreader.view;  
//import ...  
public class ScreenReaderView extends BorderPane {  
    private Button readButton;  
    private TextArea textArea;  
  
    public ScreenReaderView() {  
        readButton = new Button("Read Aloud");  
        textArea = new TextArea("Enter text to read");  
        setCenter(textArea);  
        setBottom(readButton);  
        BorderPane.setAlignment(readButton, Pos.CENTER);  
        BorderPane.setMargin(readButton, new Insets(10, 10, 10, 10));  
    }  
}
```

Declare UI controls as private attributes

In constructor:

- initialise controls
- layout the pane

✓ 3: Separate View class

```
package be.kdg.screenreader.view;  
//import ...  
public class ScreenReaderView extends BorderPane {  
    private Button readButton;  
    private TextArea textArea;  
  
    public ScreenReaderView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        readButton = new Button("Read Aloud");  
        textArea = new TextArea("Enter text to read");  
    }  
  
    private void layoutNodes() {  
        setCenter(textArea);  
        setBottom(readButton);  
        BorderPane.setAlignment(readButton, Pos.CENTER);  
        BorderPane.setMargin(readButton, new Insets(10, 10, 10, 10));  
    }  
}
```

In constructor:

- initialise controls
- layout the pane

Use methods to organise code

✓ 3: Call View class

M2ScreenReader/screenreader.Main

```
public class Main extends Application {  
  
    public static void main(String[] args) { launch(args); }  
  
    @Override  
    public void start(Stage primaryStage) {  
        primaryStage.setTitle("Screen Reader");  
        primaryStage.setWidth(300);  
        primaryStage.setHeight(450);  
        primaryStage.setScene(new Scene(new ScreenReaderView()));  
        primaryStage.show();  
    }  
}
```

Main class is responsible for

- launching the application
- setting the scenegraph in the scene
- setting the scene on stage



4.1: Write the class to handle the event



- Implement EventHandler <E>

```
public void handle(E event)
```

```
M2ScreenReader/screenreader.view.ScreenReaderView
```

EventHandler

```
class ReadAloudHandler implements EventHandler<ActionEvent> {  
    @Override  
    public void handle(ActionEvent event) {  
        //For now we just show an alert  
        Alert alert = new Alert(Alert.AlertType.INFORMATION);  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
    }  
}
```

Event!

4.2: attach code to Target

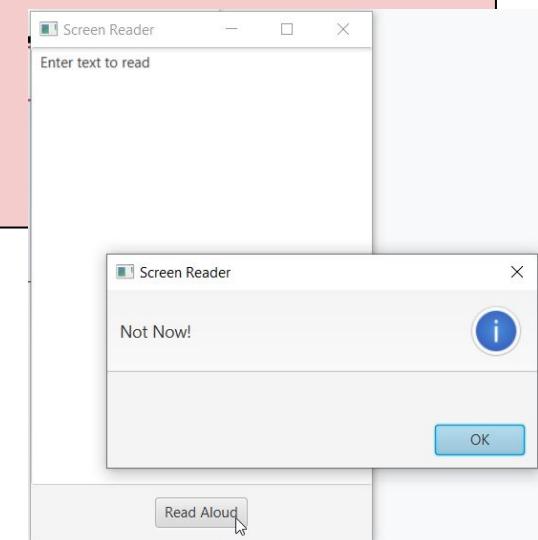
M2ScreenReader/screenreader.view.ScreenReaderView

```
public class ScreenReaderView extends BorderPane {  
    private Button readButton;  
    private TextArea textArea;  
  
    public ScreenReaderView() {  
        initialiseNodes();  
        layoutNodes();  
        addEventHandlers();  
    }  
  
    private void addEventHandlers() {  
        readButton.setOnAction(new ReadAloudHandler());  
    }  
    //...  
}
```

For each EventType, a setOn...() method exists

*readButton.*setOnAction(*new ReadAloudHandler()*);

Event Target



✓4: Handle Events

M2ScreenReader/screenreader.view.ScreenReaderView

```
package be.kdg.screenreader.view;
//import ...

public class ScreenReaderView extends BorderPane {
    private Button readButton;
    private TextArea textArea;

    class ReadAloudHandler implements EventHandler<ActionEvent> {
        @Override
        public void handle(ActionEvent event) {
            //For now we just show an alert
            Alert alert = new Alert(Alert.AlertType.INFORMATION);
            alert.setHeaderText("Not Now!");
            alert.setTitle("Screen Reader");
            alert.showAndWait();
        }
    }
    //... View constructor and other methods not shown

    private void addEventHandlers() {
        readButton.setOnAction(new ReadAloudHandler());
    }
}
```

We only use the EventHandler in the View.
ReadAloudHandler can be an inner class

Roundup

- View class: builds the UI
 - adds the eventhandler that reacts to events and changes the UI
- Main class: launches the application
 - makes the View, sets it in the Scene and shows it on the stage



- TODO:
 - Use Text To Speech (TTS) library
 - Structure the application



Lambda expressions



EventHandler is a Functional Interface ()*

```
@FunctionalInterface  
public interface EventHandler<T extends Event>  
extends EventListener
```

Handler for events of a specific class / type.

Since:

JavaFX 2.0

Method Summary

All Methods Instance Methods Abstract Methods

Modifier and Type

Method

Description

void

`handle (T event)`

Invoked when a specific event of the type for which this handler is registered happens.

The only thing a Functional Interface contains is the signature of exactly one method that must be implemented

A functional interface that is used at only one location can be written with the special lambda syntax
See the following slides!

(*) A functional interface **can contain only one abstract method**
it can additionally have static and default methods

Event Handler without lambda

M2ScreenReader/screenreader.view.ScreenReaderView

```
class ReadAloudHandler implements EventHandler<ActionEvent> {  
    public void handle(ActionEvent event) {  
        Alert alert = new Alert(Alert.AlertType.INFORMATION);  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
    }  
}  
  
private void addEventHandlers() {  
    readButton.setOnAction(new ReadAloudHandler());  
}  
//other parts of View omitted...
```

Note: We only use the event handler `ReadAloudHandler` once,
as a parameter to `readButton.setOnAction(...)`

Event Handler with -> lambda

```
class ReadAloudHandler implements EventHandler<ActionEvent> {  
    public void handle(ActionEvent event) {  
        Alert alert = new Alert(Alert.AlertType.INFORMATION);  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
    }  
  
    private void addEventHandlers() {  
        readButton.setOnAction(new ReadAloudHandler());  
    }  
    //other parts of View omitted...
```

```
private void addEventHandlers() {  
    readButton.setOnAction((event) -> {  
        Alert alert = new Alert(Alert.AlertType.INFORMATION);  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
    });  
}
```

-> special lambda syntax

Event Handler with lambda

- We only use the event handler once, as a parameter to `readButton.setAction(...)`
 - write the eventhandler lambda at the location of the `setAction(...)` parameter
- There is only one method in the interface, so Java knows its **name** (`handle`) and its parameter types (1 parameter of type `ActionEvent`)
 - You do not need to put a method name or parameter type in the lambda

```
private void addEventHandlers() {  
    readButton.setOnAction(I(event) -> {  
        Alert alert = new Alert(Alert.AlertType.INFORMATION) ;  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
I);  
}
```

Event Handler with lambda: simplification

- If the lambda has **only one parameter**, **(event)**, you may omit the round brackets around it: **event**
- A lambda is nothing but a compact notation: the code does **exactly the same as the original code with the ReadAloudHandler inner class**

M2ScreenReader/screenreaderlambda.view.ScreenReaderView

```
private void addEventHandlers() {  
    readButton.setOnAction(event -> {  
        Alert alert = new Alert(Alert.AlertType.INFORMATION);  
        alert.setHeaderText("Not Now!");  
        alert.setTitle("Screen Reader");  
        alert.showAndWait();  
    });  
}
```

Complete View class with lambda

M2ScreenReader/screenreaderlambda.view.ScreenReaderView

```
public class ScreenReaderView extends BorderPane {  
    private Button readButton;  
    private TextArea textArea;  
  
    public ScreenReaderView() {  
        initialiseNodes();  
        layoutNodes();  
        addEventHandlers();  
    }  
  
    private void addEventHandlers() {  
        readButton.setOnAction( event -> {  
            Alert alert = new Alert(Alert.AlertType.INFORMATION);  
            alert.setHeaderText("Not Now!");  
            alert.setTitle("Screen Reader");  
            alert.showAndWait();  
        });  
    }  
  
    private void initialiseNodes() { ... }  
    private void layoutNodes() { ... }  
}
```

Another lambda example: Predicate

Module java.base

Package java.util.function

Interface Predicate<T>

Type Parameters:

T - the type of the input to the predicate

Functional Interface:

This is a functional interface and can therefore be used as the assignment target for a lambda expression or method reference.

```
@FunctionalInterface  
public interface Predicate<T>
```

Represents a predicate (boolean-valued function) of one argument.

This is a functional interface whose functional method is `test(Object)`.

Method Summary

All Methods	Static Methods	Instance Methods	Abstract Methods	Default Methods
Modifier and Type	Method			Description
default Predicate<T>	<code>and(Predicate<? super T> other)</code>			Returns a composed predicate that represents a short-circuiting logical AND of this predicate and another.
static <T> Predicate<T>	<code>isEqual(Object targetRef)</code>			Returns a predicate that tests if two arguments are equal according to <code>Objects.equals(Object, Object)</code> .
default Predicate<T>	<code>negate()</code>			Returns a predicate that represents the logical negation of this predicate.
static <T> Predicate<T>	<code>not(Predicate<? super T> target)</code>			Returns a predicate that is the negation of the supplied predicate.
default Predicate<T>	<code>or(Predicate<? super T> other)</code>			Returns a composed predicate that represents a short-circuiting logical OR of this predicate and another.
boolean	<code>test(T t)</code>			Evaluates this predicate on the given argument.

Another lambda example: Predicate

M3Lambda/person.Person

```
public class Person {  
    private int age;  
    private String name;  
    private char sex;  
  
    public Person(String name, int age, char sex) {  
        this.age = age;  
        this.name = name;  
        this.sex = sex;  
    }  
  
    public int getAge() {return age; }  
  
    public void setAge(int age) {this.age = age; }  
  
    public String getName() {return name; }  
  
    public void setName(String name) {this.name = name; }  
  
    public char getSex() {return sex; }  
  
    public void setSex(char sex) { this.sex = sex; }  
  
    @Override  
    public String toString() {  
        return "Person{" + "age=" + age + ", name='" + name + '\'' +  
               ", sex=" + sex + '}';  
    }  
}
```

Another lambda example: Predicate

M3Lambda/person.PersonRepository

```
import java.util.ArrayList;
import java.util.List;
import java.util.function.Predicate;

public class PersonRepository {
    private final List <Person> data = new ArrayList<>();

    public List<Person> getBy(Predicate<Person> predicate){
        List <Person> result = new ArrayList<>();
        for (Person person : data) {
            if (predicate.test(person)) {
                result.add(person);
            }
        }
        return result;
    }

    public void add(Person person){
        data.add(person);
    }
}
```

Implementing the Predicate interface

M3Lambda/person.Main

```
public class Main {  
    public static void main(String[] args) {  
        PersonRepository database = new PersonRepository();  
        database.add(new Person("Angele", 27, 'F' ));  
        database.add(new Person("Romeo Elvis", 30, 'M' ));  
        database.add(new Person("Marka", 60, 'M' ));  
        database.add(new Person("Lisette", 1, 'F' ));  
  
        System.out.println("Printing all adults: " +  
            database.getBy(new AdultTester()));  
        //...  
    }  
}
```

Printing all adults: [Person{age=27, name='Angele', sex=F}, Person{age=30, name='Romeo Elvis', sex=M}, Person{age=60, name='Marka', sex=M}]

M3Lambda/person.AdultTester

```
import java.util.function.Predicate;  
public class AdultTester implements Predicate<Person> {  
    @Override  
    public boolean test(Person person) {  
        return person.getAge() > 18;  
    }  
}
```

Predicate with lambda's

M3Lambda/person.Main

```
public class Main {  
    public static void main(String[] args) {  
        //...  
        System.out.println("Printing all adults: " +  
            database.getBy(new AdultTester()));  
  
        System.out.println("Printing all adults: " +  
            database.getBy(person -> {return person.getAge() > 18;}));  
        System.out.println("Printing all ladies: " +  
            database.getBy(person -> {return person.getSex() == 'F';}));  
    }  
}
```

Printing all ladies: [Person{age=27, name='Angele', sex=F}, Person{age=1, name='Lisette', sex=F}]

classic syntax

lambda

M3Lambda/person.AdultTester

```
import java.util.function.Predicate;  
public class AdultTester implements Predicate<Person> {  
    @Override  
    public boolean test(Person person) {  
        return person.getAge() > 18;  
    }  
}
```

Predicate with lambda's: simplification

M3Lambda/person.Main

```
public class Main {  
    public static void main(String[] args) {  
        //...  
        System.out.println("Printing all adults: " +  
            database.getBy(new AdultTester()));  
        System.out.println("Printing all ladies: " +  
            database.getBy(p -> {return p.getSex() == 'F';}));  
        System.out.println("Printing persons with multi word names: " +  
            database.getBy(p -> p.getName().contains(" ")));  
    }  
    If a lambda only contains a return statement  
    You may drop {return ... ;}  
}
```

Printing persons with multi word names:
[Person{age=30, name='Romeo Elvis', sex=M}]

M3Lambda/person.AdultTester

```
import java.util.function.Predicate;  
public class AdultTester implements Predicate<Person> {  
    @Override  
    public boolean test(Person person) {  
        return person.getAge() > 18;  
    }  
}
```

Predicate with lambda's: removeIf example

- *removeIf*-method in the interface *Collection*:
Removing elements matching a Predicate from a Collection *without using an iterator*

default boolean `removeIf(Predicate<? super E> filter)` Removes all of the elements of this collection that satisfy the given predicate.

M3Lambda/removeIf.Eraser

```
import java.util.*;  
  
public class Eraser {  
  
    public static void main(String[] args) {  
        Collection<String> band= new ArrayList<>(  
            List.of("Dave", "Dee", "Dozy", "Beaky", "Mick", "Titch"));  
        band.removeIf(member -> member.endsWith("y"));  
        System.out.println("Members not ending with y: " + band);  
    }  
}
```

Members not ending with y: [Dave, Dee, Mick, Titch]

The model

Separation of concerns

- It is recommended to keep domain data and business logic separate from the user interface
 - allows you to easily use different interfaces (ascii, desktop, Web...)
- Put these in the `be.kdg.screenreader.model` package

M2ScreenReader/screenreader/ScreenReader

```
package be.kdg.screenreader5.model;

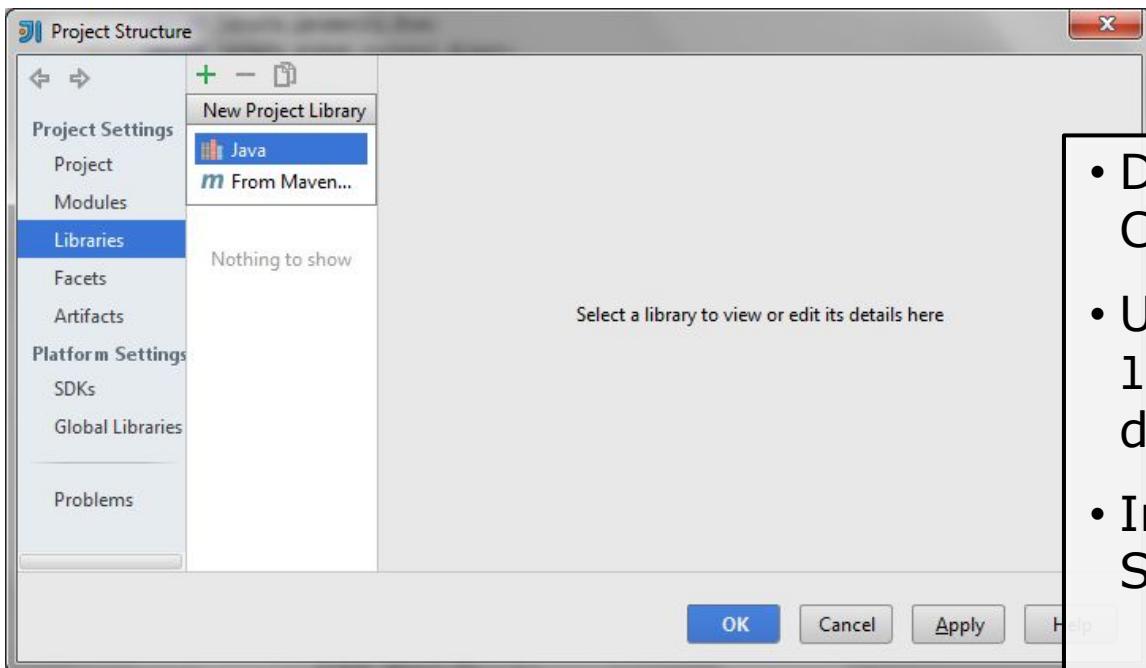
public class ScreenReader {
    private String text = "No text";

    public void setText(String text) {this.text = text;}

    public String getText() {return text;}

}
```

FreeTextToSpeech library



M2ScreenReader/src/module-info.java

```
open module M2ScreenReader {  
    requires javafx.controls;  
    requires javafx.media;  
    requires freetts;  
}
```

- Download `freetts.zip` from Canvas
- Unzip `freetts.zip` to a folder `lib/freetts` in your project directory
- In IntelliJ File | Project Structure:
 - add a new Java project library
 - add all jars in folder `freetts` to it
 - name your library `freetts`
- Add the library to `module-info.java`

model.ScreenReader

M2ScreenReader/screenreader/ScreenReader

```
package be.kdg.screenreader.model;
import com.sun.speech.freetts.Voice;
import com.sun.speech.freetts.VoiceManager;

public class ScreenReader {
    private String voiceName;
    private String text = "No text";

    public ScreenReader() {
        System.setProperty("freetts.voices",
            "com.sun.speech.freetts.en.us.cmu_us_kal.Ke
        this.voiceName = "kevin16";
    }
    public void setText(String text) {this.text = text;}
    public String getText() {return text;}

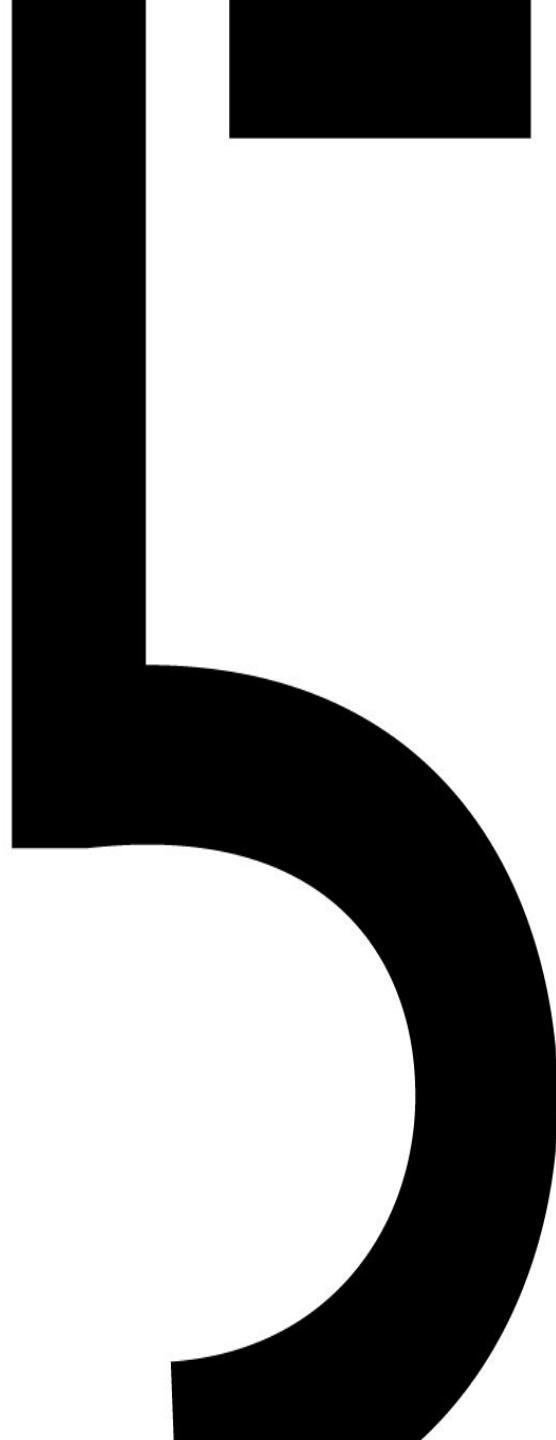
    public void readAloud() {
        Voice voice = VoiceManager.getInstance().getVoice(voiceName);
        voice.allocate();
        voice.speak(text);
        voice.deallocate();
    }
}
```

Configures FreeTTS engine

Try out the model

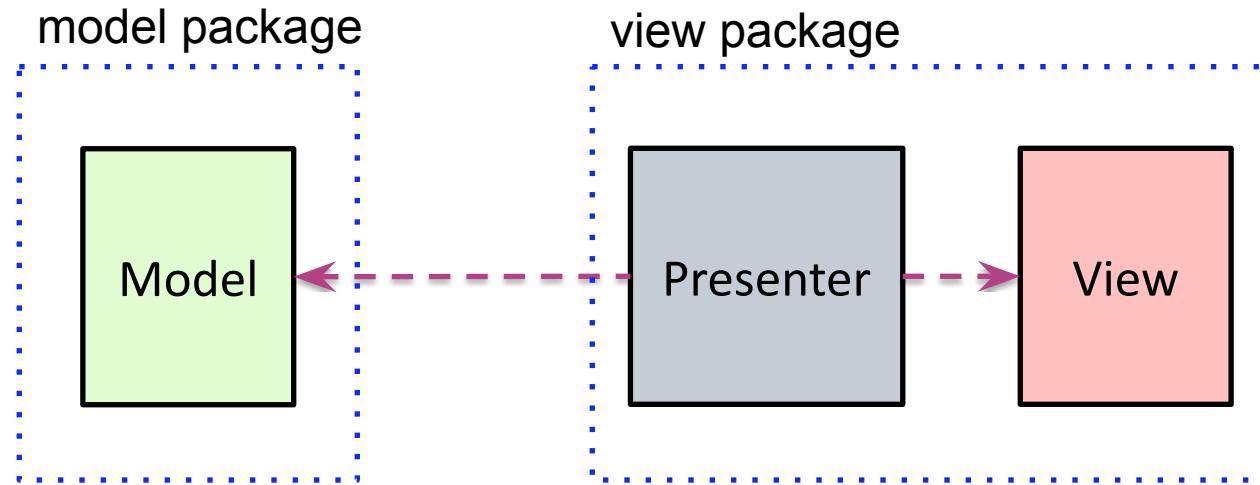
- Add a `ModelTest` class with a main method that
 - creates a `ScreenReader`
 - sets a text
 - reads it

**Model
View
Presenter**



Model View Presenter (MVP)

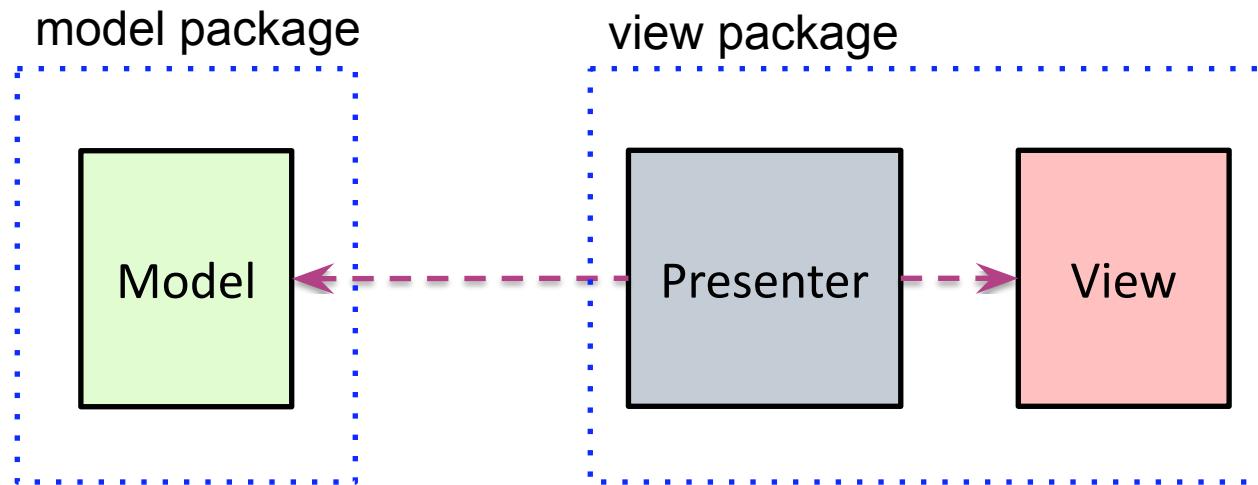
- We will always use the same pattern:
 - View classes: UI, (almost) no logic (pretty but dumb)
 - Model classes: logic, no UI (smart but ugly)
 - Presenter: Model and view are separated. The presenter connects the two (matchmaker)



MVP in ScreenReader



- **View class:** ScreenReaderView.java
→ in be.kdg.screenreader.view.screenreaderview
- **Model class:** ScreenReader.java
→ in be.kdg.screenreader.model
- **Presenter:** ScreenReaderPresenter.java
→ in be.kdg.screenreader.view.screenreaderview



Presenter

M2ScreenReader/screenreader/view/ScreenPresenter

```
public class ScreenReaderPresenter {  
    private final ScreenReader model;  
    private final ScreenReaderView view;  
  
    public ScreenReaderPresenter(  
        ScreenReader model, ScreenReaderView view) {  
        this.model = model;  
        this.view = view;  
        addEventHandlers();  
        updateView();  
    }  
    private void updateView() {  
        view.getTextArea().setText(model.getText());  
    }  
  
    private void addEventHandlers() {  
        view.getReadButton().setOnAction( event -> {  
            model.setText(view.getTextArea().getText());  
            model.readAloud();  
            updateView();  
        } );  
    }  
}
```

1. constructor receives model and view and saves them as attributes
2. updateView fills view with data from model
3. addEventHandlers reacts to view events by calling methods on model (moved here from View)

EventHandler lambda

View

M2ScreenReader/screenreader/view/ScreenView

```
public class ScreenReaderView extends BorderPane {  
    private Button readButton;  
    private TextArea textArea;  
  
    public ScreenReaderView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
    private void layoutNodes() {  
        setCenter(textArea);  
        setBottom(readButton);  
        BorderPane.setAlignment(readButton, Pos.CENTER);  
        BorderPane.setMargin(readButton, new Insets(10, 10, 10, 10));  
    }  
    private void initialiseNodes() {  
        readButton = new Button("Read Aloud");  
        textArea = new TextArea("Enter text to read");  
    }  
  
    Button getReadButton() { return readButton; }  
    TextArea getTextArea() { return textArea; }  
}
```

- View only draws
- EventHandlers moved to Presenter.
- Getters:
 - provide access for Presenter
 - package-private: restricted access, but OK for Presenter (in view package)

Model

M2ScreenReader/screenreader/model/ScreenReader

```
public class ScreenReader {  
    private String voiceName;  
    private String text = "No text";  
  
    public ScreenReader() {  
        System.setProperty("freetts.voices",  
            "com.sun.speech.freetts.en.us.cmu_us_kal.KevinVoiceDirectory");  
        this.voiceName = "kevin16";  
    }  
    public void setText(String tekst) {this.text = tekst;}  
  
    public String getText() {return text;}  
  
    public void readAloud() {  
        Voice voice = VoiceManager.getInstance().getVoice(voiceName);  
        voice.allocate();  
        voice.speak(text);  
        voice.deallocate();  
    }  
}
```

- private data
- getters/setters
- methods with business logic

Main class

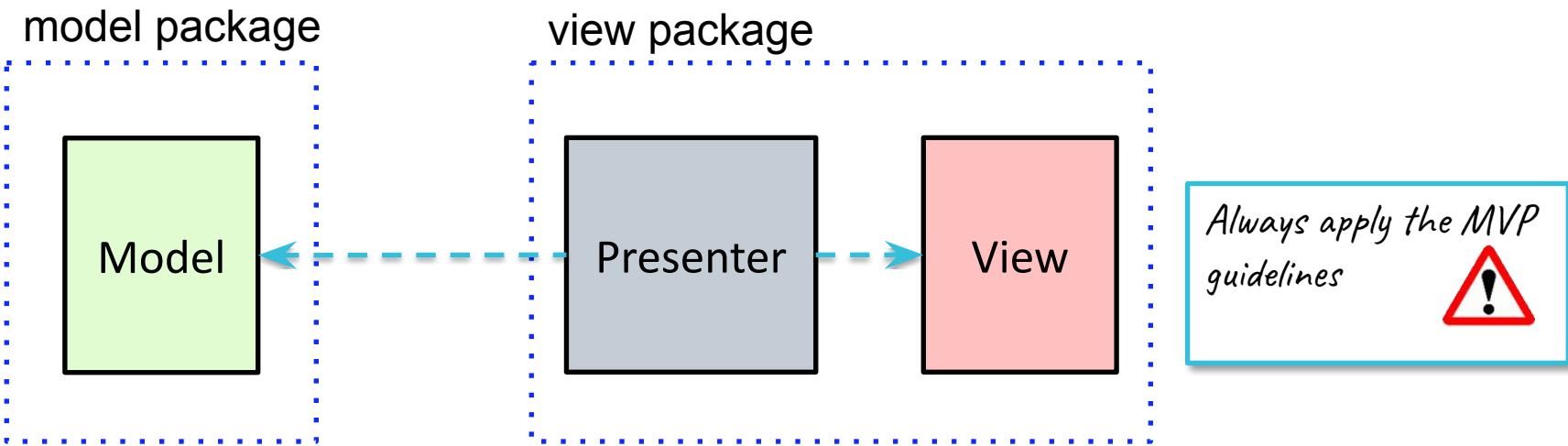
M2ScreenReader/screenreader/Main

```
public class Main extends Application {  
  
    public static void main(String[] args) {  
        launch(args);  
    }  
  
    @Override  
    public void start(Stage primaryStage) {  
        ScreenReader model = new ScreenReader();  
        model.setText("What can I say?");  
        ScreenReaderView view = new ScreenReaderView();  
  
        primaryStage.setTitle("Screen Reader");  
        primaryStage.setWidth(300);  
        primaryStage.setHeight(450);  
        primaryStage.setScene(new Scene(view));  
        ScreenReaderPresenter presenter =  
            new ScreenReaderPresenter(model, view);  
        primaryStage.show();  
    }  
}
```

1. Create the model
2. Create the View
3. Set up Stage and Scene
4. Put the view on the Scene
5. Create the presenter with model and view as parameters

MVP summary

- Model:
 - Data and logic
- Presenter:
 - Gets model and view via constructor and saves them as attributes
 - Private method addEventHandlers
 - Private method updateView
- View:
 - UI code
 - Controls are private attributes with package private getters
 - Private methods: initialiseNodes en layoutNodes



MVP Cheat Sheet version 1



Get me on Canvas!

Model View Presenter (MVP) v.1

Package applicationname

Main.java

```
package applicationname;  
import ...  
  
public class Main extends Application {  
    @Override  
    public void start(Stage primaryStage) {  
        ApplicationNameModel model =  
            new ApplicationNameModel();  
        ApplicationNameView view =  
            new ApplicationNameView();  
        new ApplicationNameView();  
        new ApplicationNameView();  
        primaryStage.setScene(new Scene(view));  
        primaryStage.show();  
    }  
    public static void main(String[] args) {  
        Application.launch(args);  
    }  
}
```

Package applicationname.model

ApplicationNameModel.java

```
package applicationname.model;  
import ...  
  
public class ApplicationNameModel {  
    // private attributes  
    public ApplicationNameModel() {  
        // Constructor  
    }  
    // methods with business logic  
    // needed getters and setters  
}
```

Package
applicationname.view

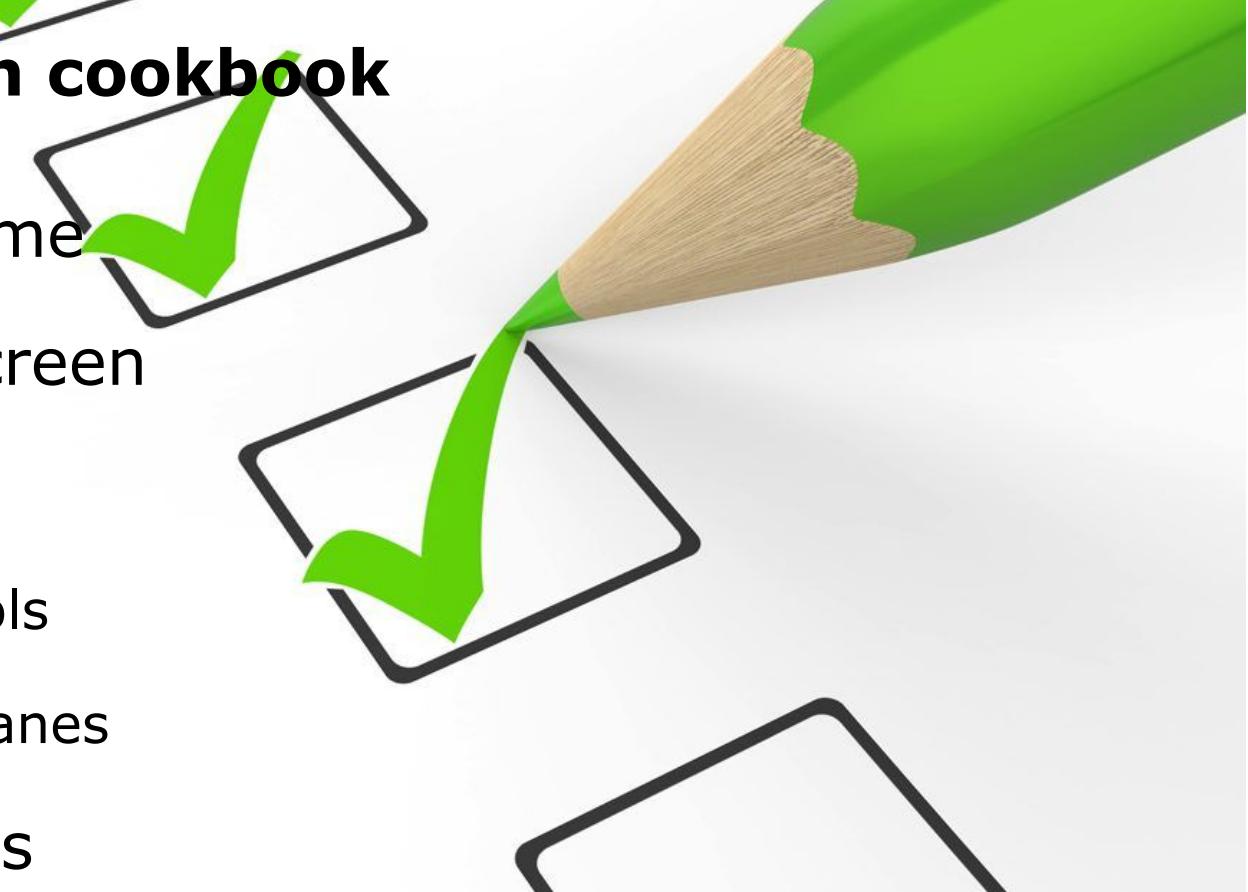
ApplicationNamePresenter.java

```
package applicationname.view;  
import ...  
  
public class ApplicationNamePresenter {  
    private ApplicationNameModel model;  
    private ApplicationNameView view;  
    public ApplicationNamePresenter(  
        ApplicationNameModel model,  
        ApplicationNameView view) {  
        this.model = model;  
        this.view = view;  
        addEventHandlers();  
        updateView();  
    }  
    private void addEventHandlers() {  
        // Adds event handlers (inner classes or  
        // Lambdas to view controls  
        // Event handlers: call model methods and  
        // update the view.  
    }  
    private void updateView() {  
        // fills the view with model data  
    }  
}
```

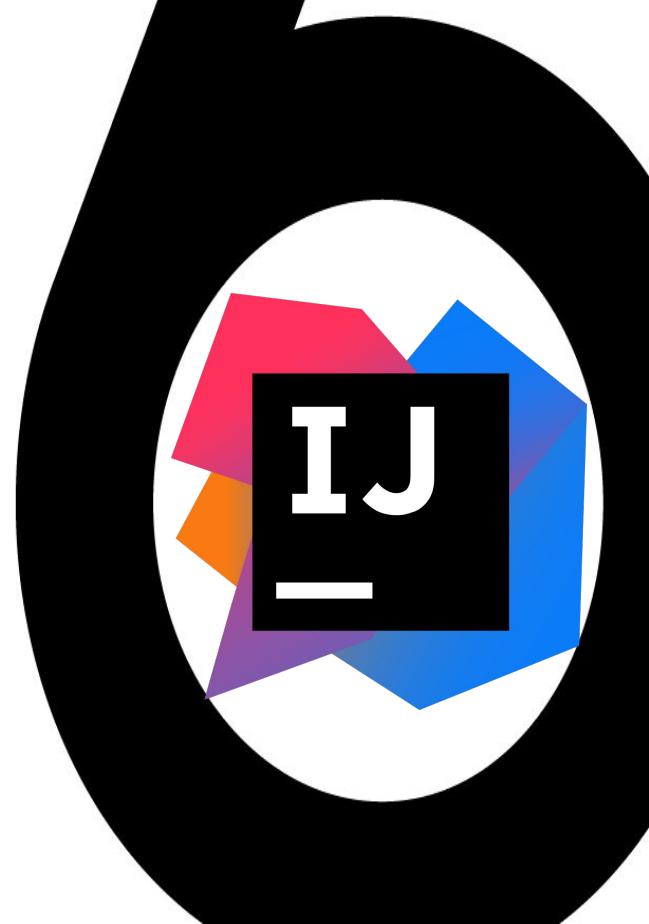
ApplicationNameView.java

```
package applicationname.view;  
import ...  
  
public class ApplicationNameView extends  
    /* layout type */ {  
    public ApplicationNameView () {  
        initializeNodes();  
        layoutNodes();  
    }  
    private void initializeNodes() {  
        // create and configure controls  
        // button = new Button("...")  
        // label = new Label("...")  
    }  
    private void layoutNodes() {  
        // add/set - methods  
        // Insets, padding, alignment, -  
        // package-private getters  
        // for controls used by Presenter  
    }  
}
```

GUI application cookbook

- 
1. Draw wireframe
 2. Build main screen
 3. Build UI
 - i. Initialise controls
 - ii. Layout using panes
 4. Handle Events
 - i. Write code to handle the event (do action, change GUI...)
 - ii. Attach code to event target (GUI control)
 5. Apply the MVP pattern

**Make an MVP
template project**



Make an MVP template project

- Apply cheat sheet (V1) in an IntelliJ template project.
- Use the template project as a starting point for all your MVP projects
- We'll set up the template project step by step

Model View Presenter (MVP) v.1

Package applicationname;

```
>Main.java
package applicationname;
import ...;

public class Main extends Application {
    public void start(Stage primaryStage) {
        primaryStage.setTitle("Hello JavaFX");
        new ApplicationNameView();
        new ApplicationNameModel();
        new ApplicationNamePresenter();
    }
    public static void main(String[] args) {
        application.launch(args);
    }
}
```

Package applicationname.view;

```
ApplicationNamePresenter.java
package applicationname.view;
import ...;

public class ApplicationNamePresenter {
    private ApplicationNameModel model;
    private ApplicationNameView view;
    public ApplicationNamePresenter() {
        model = new ApplicationNameModel();
        view = new ApplicationNameView();
        view.setPresenter(this);
    }
    private void startNewWindow() {
        // add event handlers (inner classes are used)
        // event handlers call model methods and
        // update view
    }
    private void updateView() {
        // update view with model data
    }
}
```

Package applicationname.model;

```
ApplicationNameModel.java
package applicationname.model;
import ...;

public class ApplicationNameModel {
    // private attributes
    public ApplicationNameModel() {
        // ...
    }
    // methods with business logic
    // needed getters and setters
}
```

Package applicationname.presenter;

```
ApplicationNameView.java
package applicationname.view;
import ...;

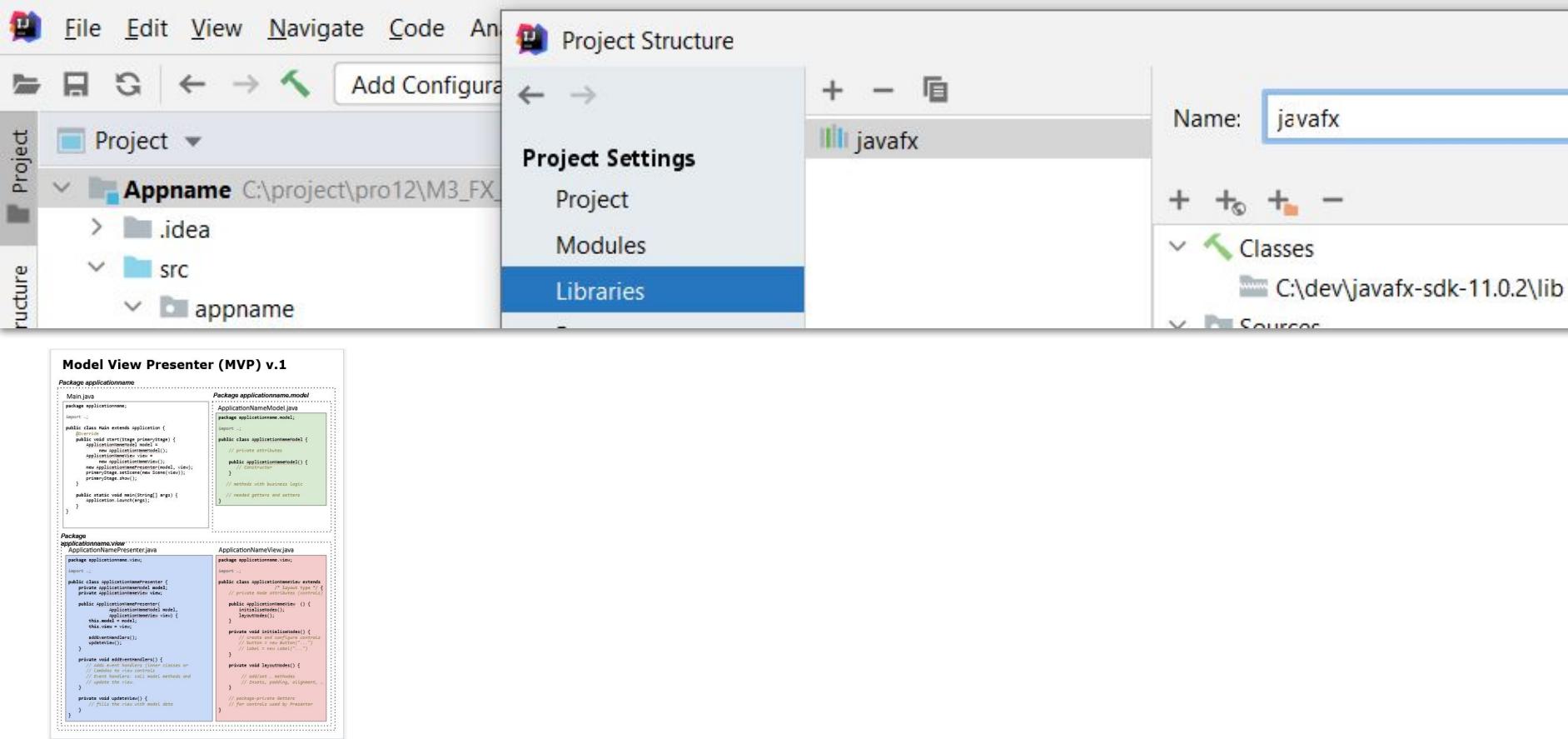
public class ApplicationNameView extends StackPane {
    // private node attributes (control)
    public ApplicationNameView() {
        initialize();
        layoutNodes();
    }
    private void initialize() {
        // create and configure controls
        // add them to view (label, " ")
    }
    private void layoutNodes() {
        // set size, methods
        // create, position, alignment, ...
    }
    // package-private methods used by presenter
    // for communication with model
}
```

Step 1. Set up the project



Exercise

- Make a new IntelliJ project called **Appname**
- Add the JavaFX libraries to the project
 - Add the module-info.java file with the javaFX modules
- Add a package **appname**

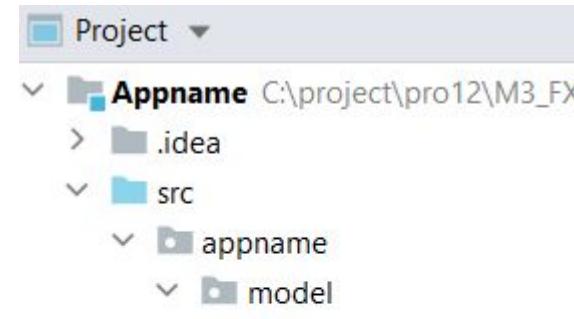


Step 2. Add the Model class



Exercise

- Add in the package *appname* the package *model*
- Add in package *model* the class *ModelName*
- Add code and comments as in the cheat sheet



```
package appname.model;

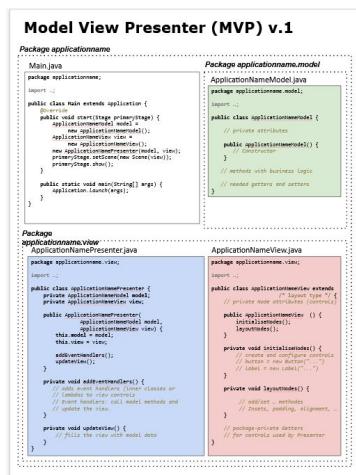
public class ModelName{

    // private attributes

    public ModelName() {
        // Constructor
    }

    // methods with business logic

    // necessary getters and setters
}
```



Step 3. Add the View class



Exercise

- Add in the package *appname* the package *view*
- Add in package *view* the class *AppNameView*
- Add code and comments as in the cheat sheet

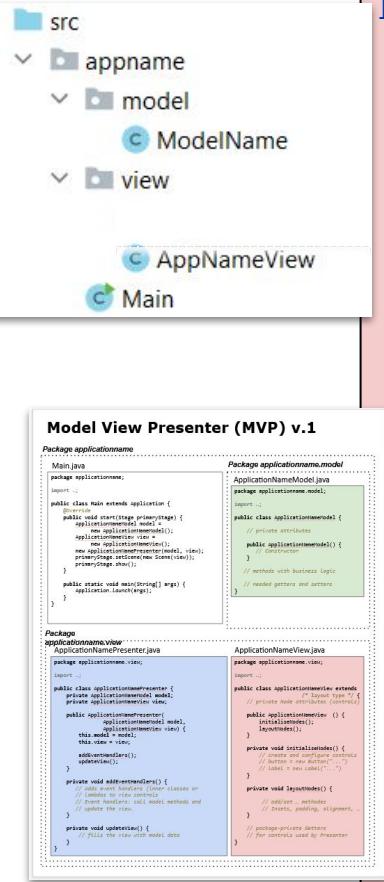
```
package appname.view;

import javafx.scene.layout.BorderPane;

public class AppNameView extends BorderPane {
    // private Node attributes (controls)
    public AppNameView () {
        initialiseNodes();
        layoutNodes();
    }

    private void initialiseNodes() {
        // create and configure controls
    }

    private void layoutNodes() {}
    // package-private Getters
    // for controls used by Presenter
}
```

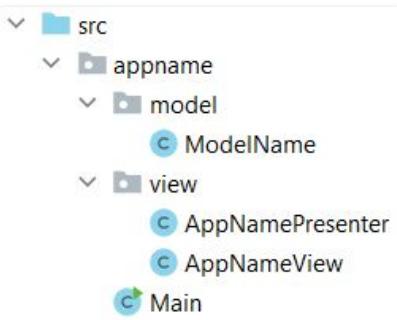


Step 4. Add the Presenter class



Exercise

- Add in package *view* the class *AppNamePresenter*
- Add code and comments as in the cheat sheet



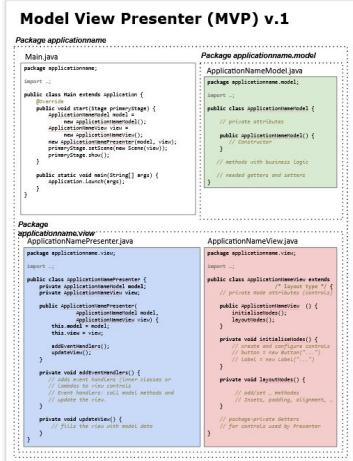
```
package appname.view;
import appname.model.ModelName;

public class AppNamePresenter {
    private final ModelName model;
    private final AppNameView view;

    public AppNamePresenter(
        ModelName model, AppNameView view) {
        this.model = model;
        this.view = view;
        addEventHandlers();
        updateView();
    }

    private void addEventHandlers() { }

    private void updateView() /* fills view */ { }
```



Step 5. Add the Main class

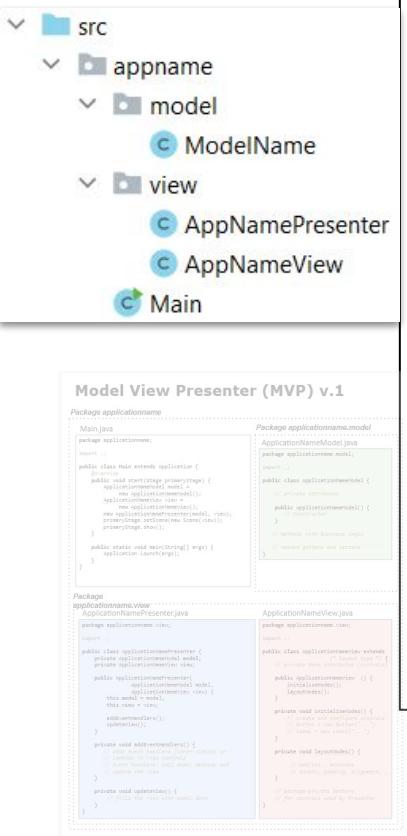
- In package `appname` create a new class called `Main`
- Use the cheatsheet to add code

//imports...

```
public class Main extends Application {

    public static void main(String[] args) {launch(args);}

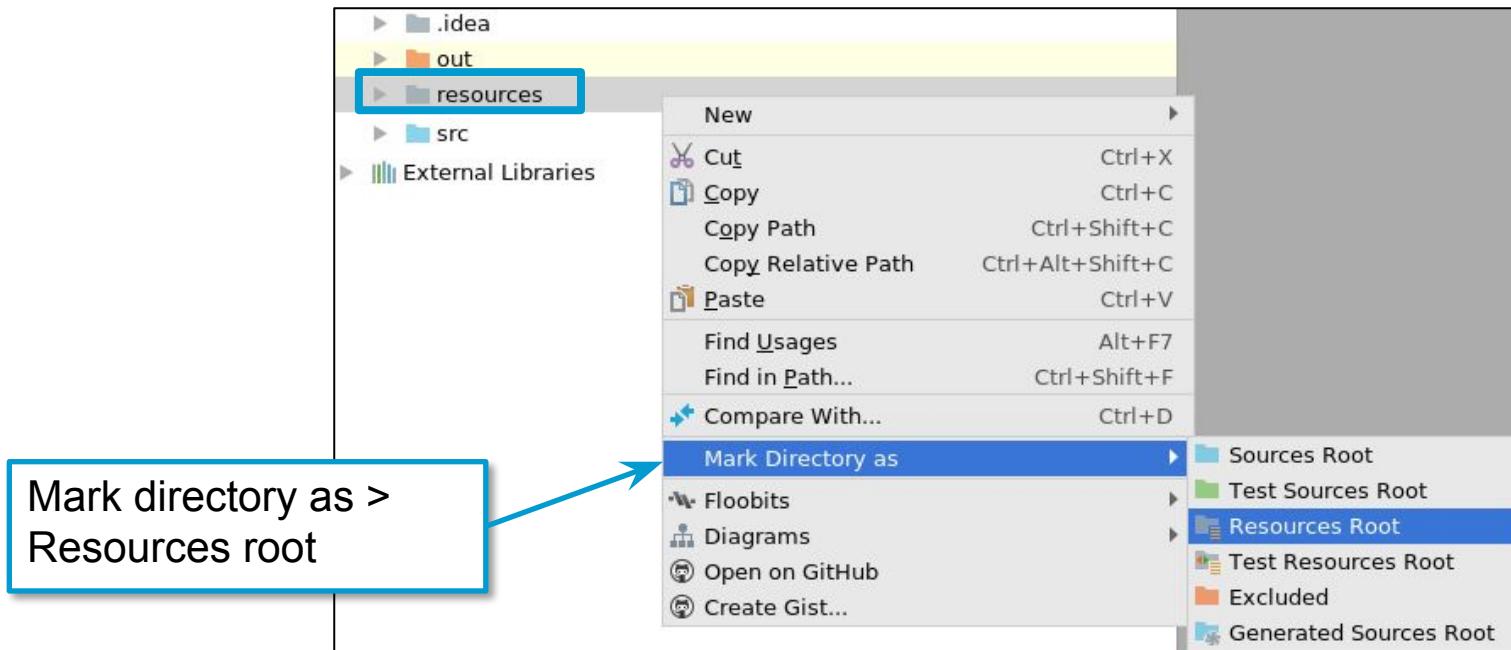
    @Override
    public void start(Stage primaryStage) {
        ModelName model = new ModelName();
        AppNameView view = new AppNameView();
        primaryStage.setScene(new Scene(model, view));
        new AppNamePresenter(model, view);
        primaryStage.show();
    }
}
```



Step 6. Add resources



- Add a resources directory to the top level of your project
 - mark the directory as resources root
 - Add a file README.md to the resources directory

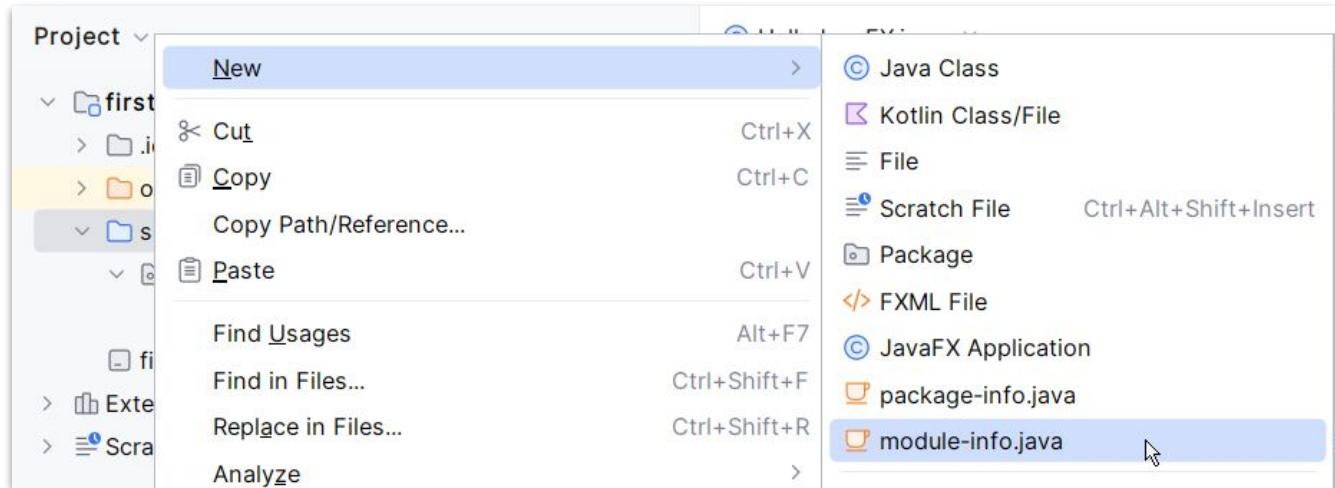


Step 7. Add a module-info.java file



Exercise

- Add a module-info.java file to the src directory



- Add module-info.java to the src-folder
Add at least the highlighted content

module-info.java

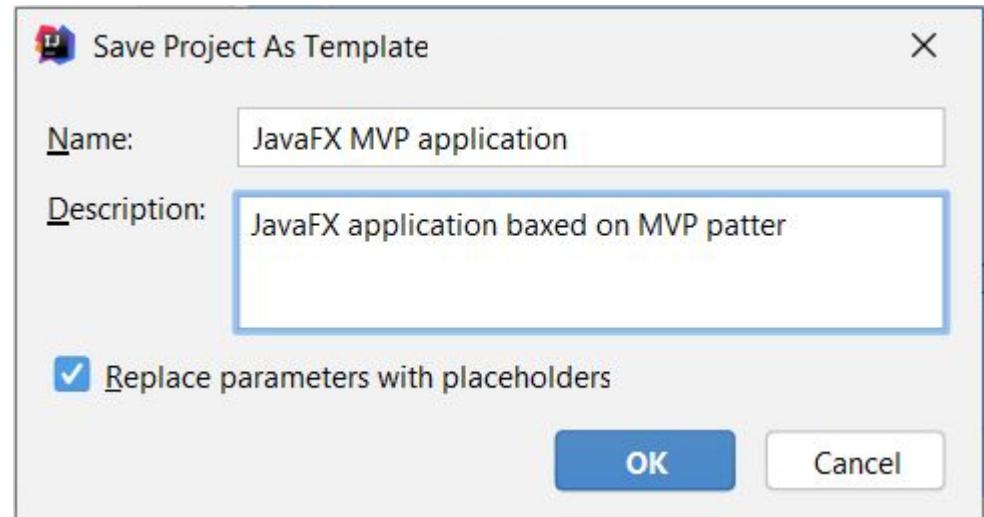
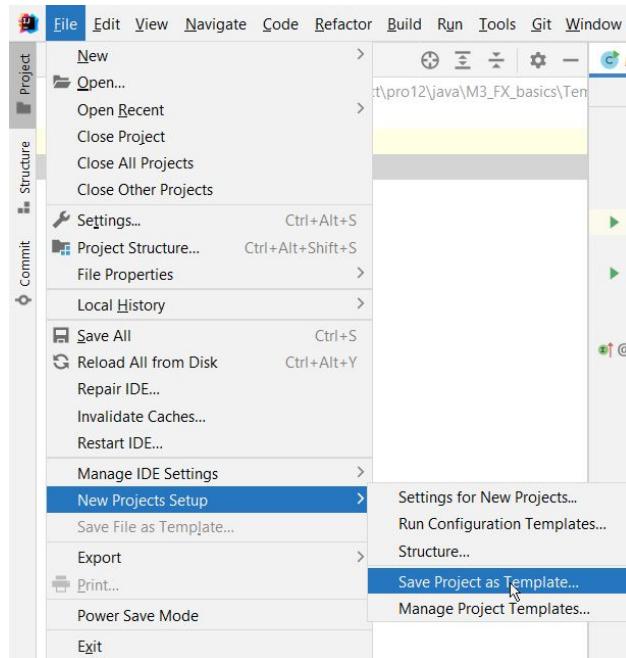
```
open module first javafx project {  
    requires javafx.controls;  
    requires javafx.media;  
    requires javafx.graphics;  
    requires javafx.base;  
}
```

Step 8. Save the project as a template



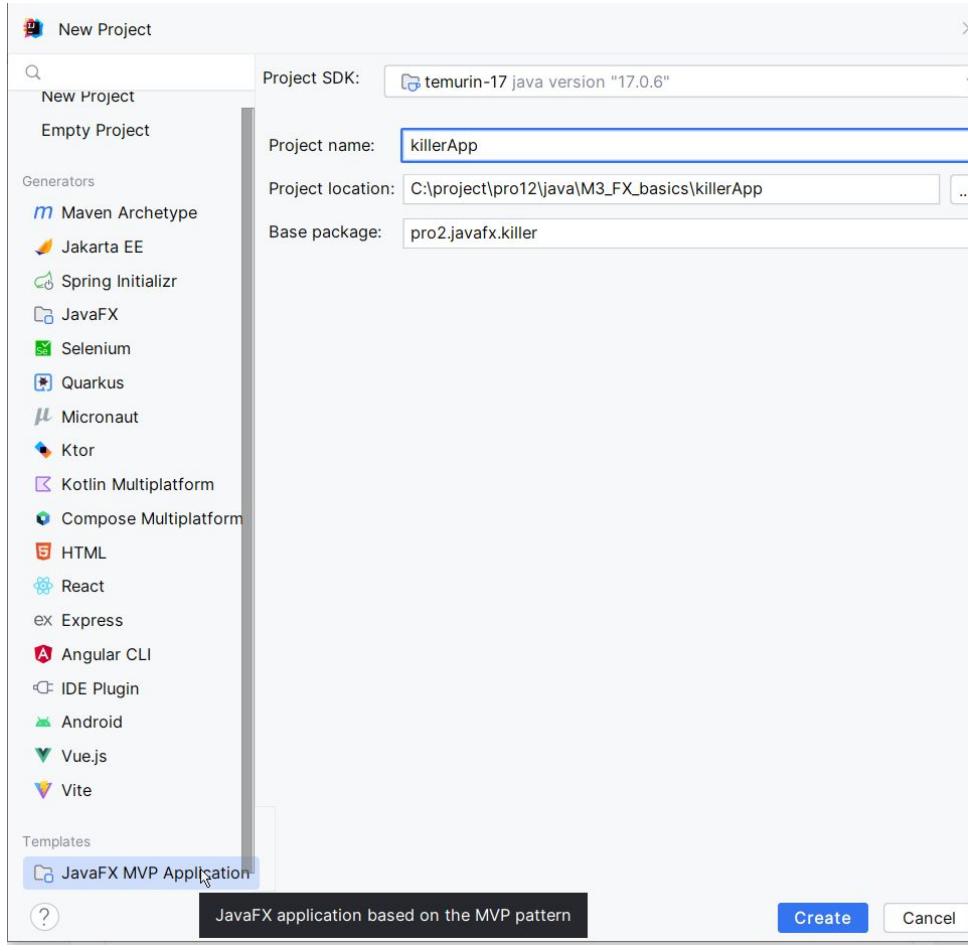
Exercise

- Verify that your project runs correctly
- If all is well, save your project as a template, using File | New Projects Setup | Save Project as Template
- Close this project



Use the project template

- When making a new JavaFX MVP project (using File | New | Project), select javaFX MVP Application under templates



Do exercises 3.02-3.05



Summary

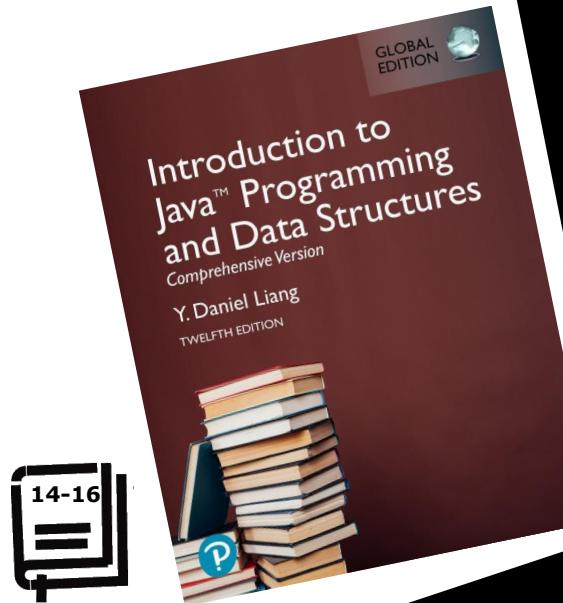
1. Node / Scene / Stage
2. A GUI application cookbook:
Screen Reader
3. Lambda expressions
4. The model
5. Model View Presenter
6. Make an MVP template project

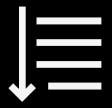
JavaFX: Nodes, Events, and Layout

Programming 2: Java



University of Applied
Sciences and Arts





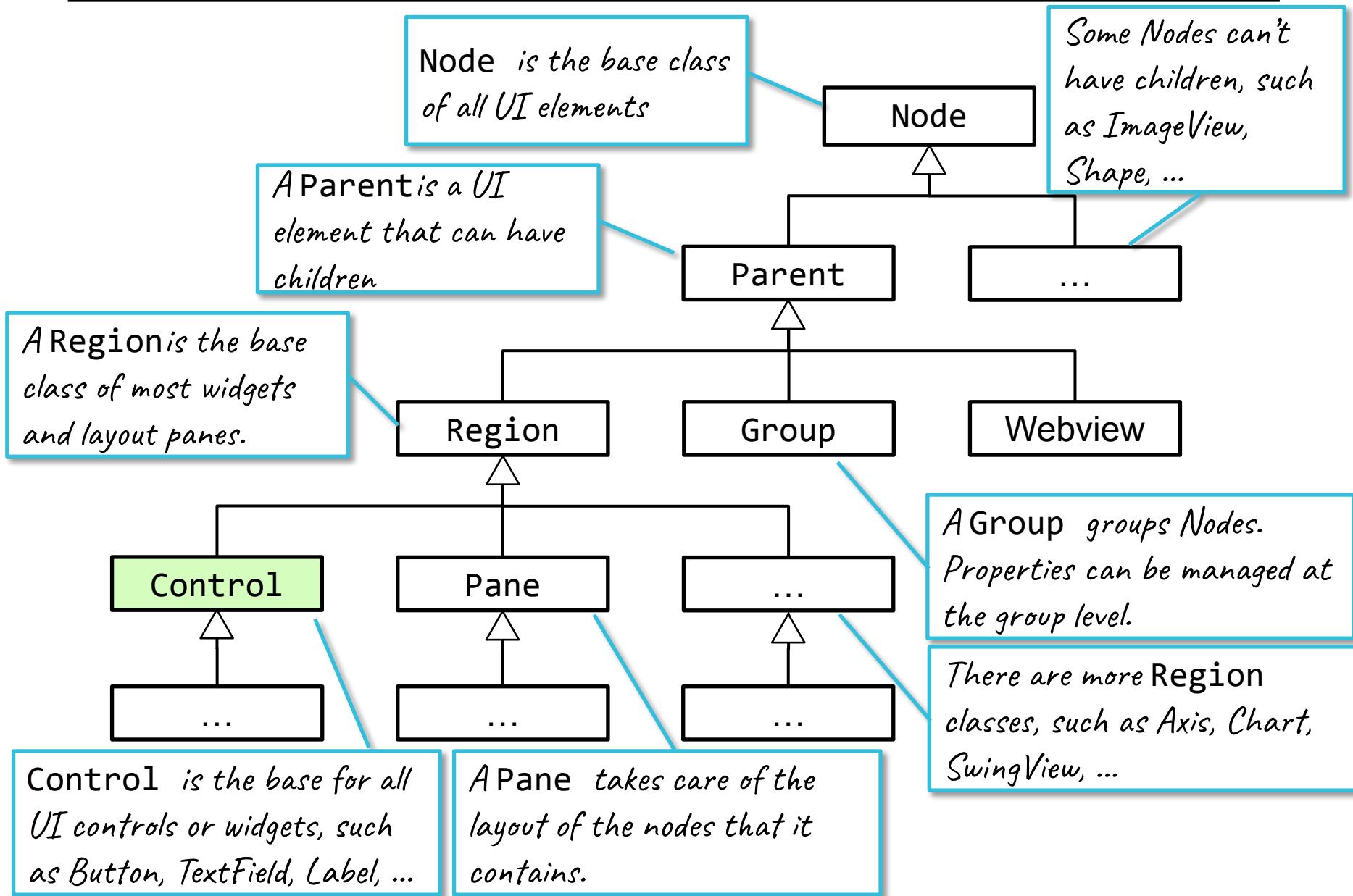
Agenda

- Controls
 - Label, ImageView, Button, CheckBox, TextField, ComboBox, Menu's
- Events
 - ActionEvent – MouseEvent – KeyEvent – WindowEvent
 - Lambdas
- Layout
 - BorderPane – HBox – VBox – GridPane
- Model - View - Presenter



Controls

Control within the class hierarchy



Controls

- In package
javafx.scene.control

- Documentation:

- https://fxdocs.github.io/docs/html5/#ui_controls

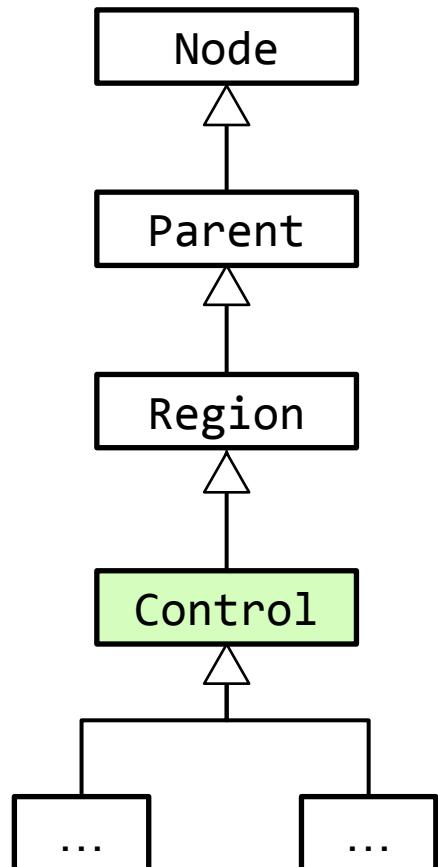
- JavaFX API documentation:

- <https://openjfx.io/javadoc/11/javafx.controls/javafx.scene/control/package-summary.html>

A screenshot of a web browser displaying the JavaFX Controls API documentation. The URL in the address bar is openjfx.io/javadoc/11/javafx.controls/javafx.scene/control/package-summary.html. The page has a header with tabs: OVERVIEW, MODULE, PACKAGE (which is selected), CLASS, USE, TREE, DEPRECATED, INDEX, and HELP. Below the header is a search bar and a table of contents. The main content area is titled "Interface Summary" and lists several interfaces: Skin<C extends Skinnable>, Skinnable, TextInputControl.Content, and Toggle. To the right of each interface is a "Description" column. The "Class Summary" section lists various classes including Accordion, Alert, Button, ButtonBar, ButtonBase, ButtonType, Cell<T>, CheckBox, CheckBoxTreeItem<T>, CheckBoxTreeItem.TreeModificationEvent<T>, CheckMenuItem, ChoiceBox<T>, ChoiceDialog<T>, ColorPicker, ComboBox<T>, ComboBoxBase<T>, ContextMenu, Control, CustomMenuItem, DateCell, and DatePicker. Each class has a corresponding "Description" column to its right.

Controls: inherited methods

- setPref/Min-maxSize/Width/Length
- setBorder
- setBackground
- setVisible
- setStyle → *apply CSS: see later*
- setTooltip
- setDisabled
- setOn... → *EventHandlers*
- ... → *Many more inherited methods: check the API documentation!*



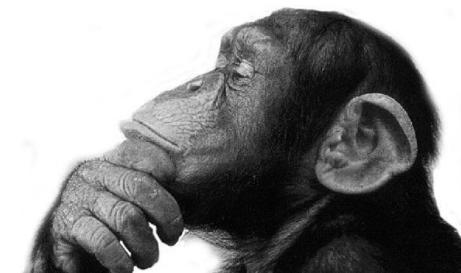


Controls: the tip of the iceberg

- Label
- Button
- TextField
- TextArea
- ComboBox
- CheckBox
- RadioButton
- List
- ColorPicker
- DatePicker
- Menu – MenuItem
- PasswordField
- ProgressBar
- RadioButton
- ScrollBar
- Slider
- Spinner
- Table
- TreeView
- ...

Do I have to
memorize all of
these?

*We'll look at a couple of them.
Look the others up in the
documentation!*





Label

```
public class LabelView1 extends BorderPane {  
    private Label label;
```

```
public LabelView1() {  
    initialiseNodes();  
    layoutNodes();  
}
```

```
private void initialiseNodes() {  
    Image imageOk = new Image("/angrybird.png");  
    label = new Label("Accept", new ImageView(imageOk));
```

```
}
```

```
private void layoutNodes() {  
    setCenter(label);  
    BorderPane.setMargin(label, new Insets(10));  
}  
}
```



You can create a Label with a text and/or an Image.
(there's also setText and setGraphics)

Label

```
public class LabelView2 extends BorderPane {  
    private Label label;
```

```
    public LabelView2() {  
        initialiseNodes();  
        layoutNodes();  
    }
```

```
    private void initialiseNodes() {  
        label = new Label("Accept");  
    }
```

```
    private void layoutNodes() {  
        label.setBackground(new Background(  
            new BackgroundFill(Color.ORANGE,  
                CornerRadii.EMPTY, Insets.EMPTY)));  
        label.setBorder(new Border(new BorderStroke(Color.RED,  
            BorderStrokeStyle.SOLID,  
            CornerRadii.EMPTY, new BorderWidths(3))));  
        label.setPrefSize(100, 50);  
        label.setAlignment(Pos.CENTER);  
        setCenter(label);  
        BorderPane.setMargin(label, new Insets(10));  
    }
```

- You can change the Background and the Border
- Text can be aligned

setBackground, setBorder,
setPrefSize...
*are inherited methods, they exist
for all controls!*



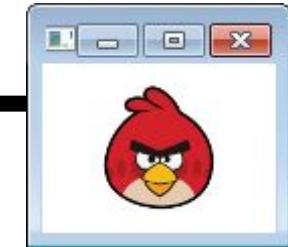
ImageView



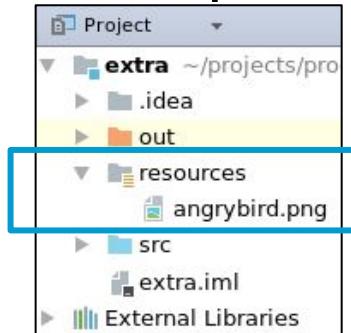
```
public class ImageViewView extends BorderPane {  
    private ImageView imageView;  
  
    public ImageViewView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        imageView = new ImageView(  
            new Image("/angrybird.png"));  
    }  
  
    private void layoutNodes() {  
        setCenter(imageView);  
        BorderPane.setMargin(imageView,  
            new Insets(10));  
    }  
}
```

ImageView doesn't have the default grey background that Label or Button has

Resources



- Images are retrieved from **resources**
 - A directory at the top level of your project



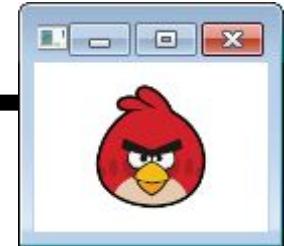
- Always mark this directory as “*Resources Root*”
- You can now load image files using
new Image (" /angrybird.png ") ;

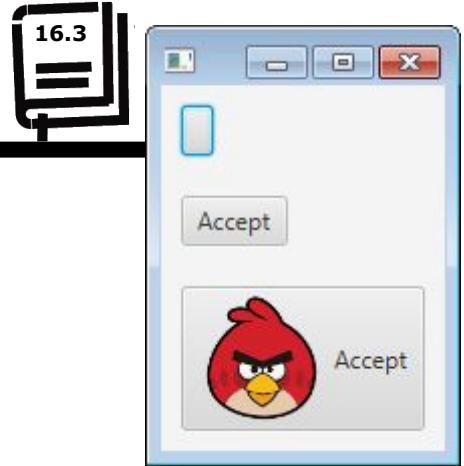
/ is the root of your resources,
so it's the directory called resources

Image

- In Java, always use a **slash** when specifying locations on a filesystem (paths), never backslash!

```
new Image("/angrybird.png");
```



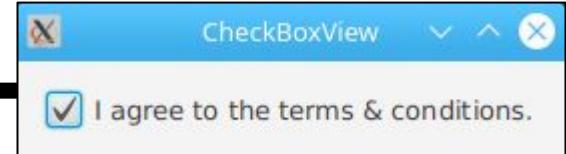


Button

```
public class ButtonView extends BorderPane {  
    private Button button1, button2, button3;  
  
    public ButtonView() {  
        initialiseNodes(); layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        button1 = new Button();  
        button2 = new Button("Accept");  
        Image imageOk = new Image("/angrybird.png");  
        button3 = new Button("Accept", new ImageView(imageOk));  
    }  
  
    private void layoutNodes() {  
        setTop(button1);  
        setCenter(button2);  
        setBottom(button3);  
        BorderPane.setMargin(button1, new Insets(10));  
        BorderPane.setMargin(button2, new Insets(10));  
        BorderPane.setMargin(button3, new Insets(10));  
        BorderPane.setAlignment(button2, Pos.CENTER_LEFT);  
    }  
}
```

can have text and/or an image

CheckBox



```
public class CheckBoxView extends BorderPane {  
    private CheckBox agree;  
  
    public CheckBoxView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        agree = new CheckBox("I agree " +  
            "to the terms & conditions.");  
    }  
  
    private void layoutNodes() {  
        setCenter(agree);  
        BorderPane.setMargin(agree,  
            new Insets(15.0));  
    }  
}
```

- can have text and/or an image
- isSelected() returns whether the CheckBox is selected

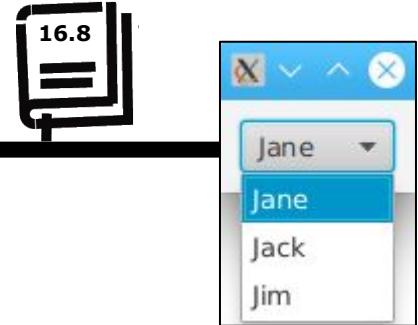
TextField



```
public class TextFieldView extends BorderPane {  
    private Label label;  
    private TextField name;  
  
    public TextFieldView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        label = new Label("Name:");  
        name = new TextField();  
    }  
  
    private void layoutNodes() {  
        BorderPane.setMargin(label, new Insets(10));  
        BorderPane.setMargin(name, new Insets(10));  
        setLeft(label);  
        setRight(name);  
    }  
}
```

- A **TextField** is an **editable text control**
- It contains a single line of text
- Obtain its content using the `getText()` method

ComboBox



```
public class ComboBoxView extends BorderPane {  
    private ComboBox<String> names;  
  
    public ComboBoxView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        names = new ComboBox<>();  
        ObservableList<String> values =  
            FXCollections.observableArrayList(  
                "Jane", "Jack", "Jim");  
        names.setItems(values);  
        names.getSelectionModel().select(0);  
    }  
  
    private void layoutNodes() {  
        setCenter(names);  
        BorderPane.setMargin(names, new Insets(10));  
    }  
}
```

ComboBox

- A ComboBox displays a list of options
- Elements are added to a ComboBox using the ObservableList class
- You can create an ObservableList out of more familiar Collections interfaces using the helper methods of the FXCollections class
- A ComboBox can be editable, allowing the user to type text.
Give it a try!

MenuBar, Menu, MenuItem

```
public class MenuBarView extends BorderPane {  
    private MenuItem exit;  
  
    public MenuBarView() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        exit = new MenuItem("Exit");  
    }  
  
    private void layoutNodes() {  
        final Menu fileMenu = new Menu("File",  
            null, exit);  
        final MenuBar menuBar = new MenuBar(fileMenu);  
        setTop(menuBar);  
        BorderPane.setMargin(menuBar,  
            new Insets(0.0, 0.0, 100.0, 0.0));  
    }  
}
```



Just the MenuItem's as attributes is fine

- can have text and/or an image
- There's also CheckMenuItem and RadioMenuItem

A Menu has zero or more MenuItem's: parameters 3 and higher!

A MenuBar has zero or more Menus.

MenuBar is the control that is placed on the Pane.

More controls...

Check this tutorial:

https://docs.oracle.com/javase/8/javafx/user-interface-tutorial/ui_controls.htm



Assignment



Complete this assignment:

→ Java 4.01: Birds

- the starter project contains the needed image





Events

Event Handling

Event is the base class of all JavaFX events.

Has getSource and consume methods, among others.

ActionEvent represents any kind of action. It's used with many kinds of buttons as well as with animations.

ActionEvent

Event

InputEvent

WindowEvent is used when a window is shown, minimized, or closed.

WindowEvent

...

MouseEvent

KeyEvent

Clicking, dragging, or moving the mouse triggers a MouseEvent.

Pressing, typing, or releasing keyboard keys triggers a KeyEvent.

Event hierarchy and methods

- In class `java.util.EventObject`:

→ `public Object getSource()`

Returns the component that triggered the event.

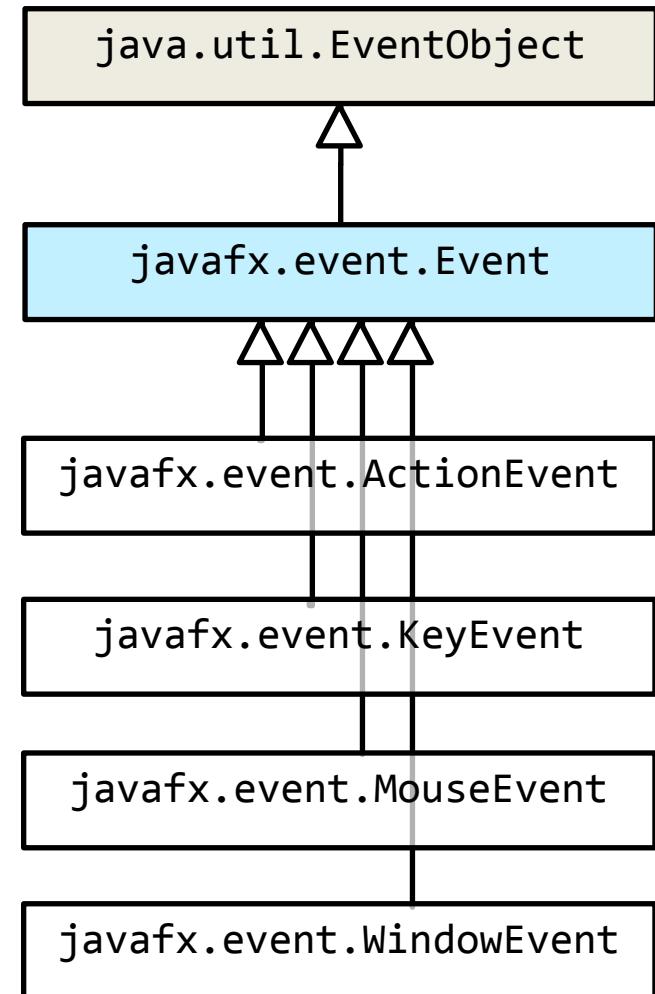
- In class `javafx.event.Event`:

→ `public void consume()`

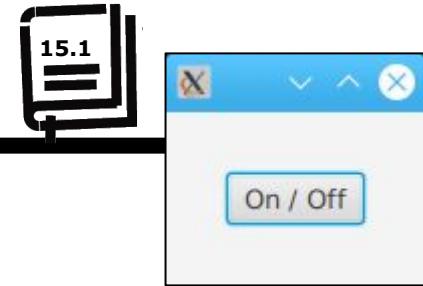
Calling this method, will cause this event not to be processed any further!

- Any Event subclass has specialised methods giving information about that type of Event

→ Example: A KeyEvent will tell you which character was entered



ActionEvent



```
public class TestActionEvent extends Application {  
    @Override  
    public void start(Stage stage) {  
        BorderPane borderPane = new BorderPane();  
        ToggleButton toggleButton = new ToggleButton("On/Off");  
        toggleButton.setOnAction(  
            event -> {  
                System.out.println(  
                    toggleButton.isSelected() ? "On" : "Off")  
            }  
        );  
        borderPane.setCenter(toggleButton);  
        BorderPane.setMargin(toggleButton, new Insets(30.0));  
        stage.setScene(new Scene(borderPane));  
        stage.show();  
    }  
}
```

We're not using MVP in these examples to keep them concise.

ActionEvent

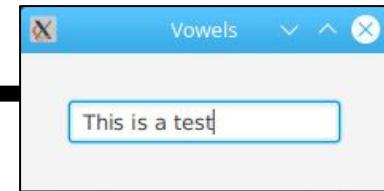
- We can add an event handler for **ActionEvent** to buttons, menu items, text fields, combo boxes, and many more. The method to use is:

→ `public void setOnAction(EventHandler<ActionEvent> handler)`

-
- Use an **inner class or lambda** that implements `EventHandler<ActionEvent>`
 - In an `EventHandler<XxxEvent>` implementation, you need to implement one method: `handle(XxxEvent e)`

We're not using MVP in these examples to keep them concise.

KeyEvent: Lambda syntax



```
public class TestKeyEvent extends Application {  
    @Override  
    public void start(Stage stage) {  
        BorderPane borderPane = new BorderPane();  
        TextField textField = new TextField();  
        textField.setOnKeyTyped(event -> {  
            if ("aeiou".contains(event.getCharacter())) {  
                System.out.println(event.getCharacter());  
            }  
        });  
  
        borderPane.setCenter(textField);  
        BorderPane.setMargin(textField, new Insets(30.0));  
        stage.setScene(new Scene(borderPane));  
        stage.setTitle("Vowels");  
        stage.show();  
    }  
}
```

Lambda's are compact and expressive.
Know exactly which interface and method
you're implementing!

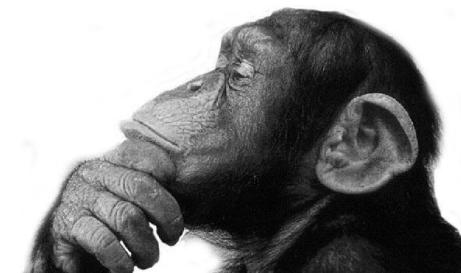
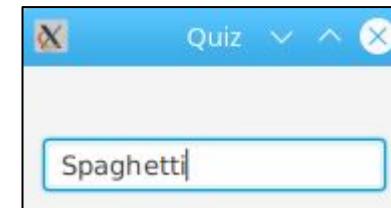
KeyEvent

- Event handlers for **KeyEvent** can be added to Nodes or Scenes.
Some methods:
 - `public void setOnKeyReleased(EventHandler<KeyEvent> handler)`
 - `public void setOnKeyTyped(EventHandler<KeyEvent> handler)`
- Use an **inner class or a lambda** that implements
`EventHandler<KeyEvent>`
 - You'll have to implement this method:
 - `public void handle(KeyEvent event)`
- Here are some of **KeyEvent**'s methods:
 - `public String getCharacter()`
 - `public KeyCode getCode()`
 - `public boolean isControlDown()`

Quiz

```
public class QuizView extends VBox {  
    public QuizView() {  
        TextField name = new TextField();  
        Label label = new Label();  
        name.setOnKeyTyped(  
            event -> label.setText(event.getCharacter()  
                + label.getText())  
        );  
        setSpacing(10);  
        setPadding(new Insets(10));  
        getChildren().addAll(label, name);  
    }  
}
```

What will appear
in this label?



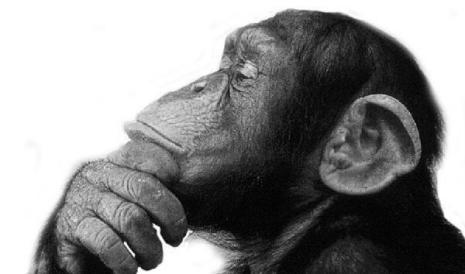
MouseEvent

```
public class TestMouseEvent extends Application {  
    @Override  
    public void start(Stage stage) {  
        BorderPane borderPane = new BorderPane();  
        borderPane.setOnMouseMoved(  
            event ->  
                System.out.printf("X: %3.0f, Y: %3.0f%n",  
                    event.getX(), event.getY())  
        );  
        stage.setScene(new Scene(borderPane));  
        stage.setTitle("Coordinates");  
        stage.show();  
    }  
}
```

Output:

```
X: 110, Y: 164  
X: 112, Y: 164  
X: 116, Y: 165  
X: 123, Y: 165  
X: 136, Y: 166  
X: 151, Y: 167  
...
```

Where's the origin
(0,0) of this
coordinate system?



MouseEvent

- Event handlers for **MouseEvent** can be added to Nodes and to Scenes. These are the methods to use:

- `public void setOnMouseClicked(EventHandler<MouseEvent> handler)`
- `public void setOnMouseEntered(EventHandler<MouseEvent> handler)`
- `public void setOnMouseExited(EventHandler<MouseEvent> handler)`
- `public void setOnMouseMoved(EventHandler<MouseEvent> handler)`
- `public void setOnMousePressed(EventHandler<MouseEvent> handler)`
- `public void setOnMouseReleased(EventHandler<MouseEvent> handler)`

... and also

- `public void setOnMouseDragEntered(EventHandler<MouseEvent> handler)`
- `public void setOnMouseDragExited(EventHandler<MouseEvent> handler)`
- `public void setOnMouseDragged(EventHandler<MouseEvent> handler)`
- `public void setOnMouseDragOver(EventHandler<MouseEvent> handler)`
- `public void setOnMouseDragReleased(EventHandler<MouseEvent> handler)`

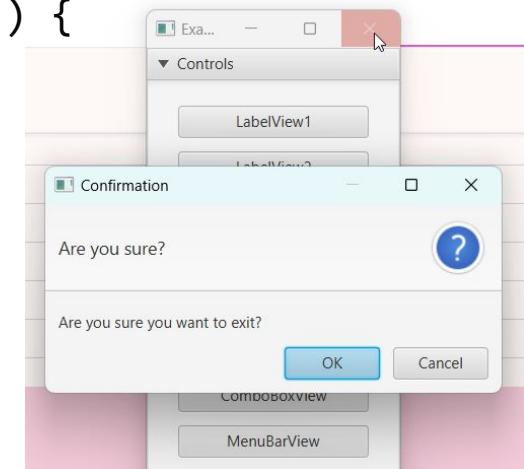
MouseEvent

- Use an **inner class or a lambda** that implements `EventHandler<MouseEvent>`
- You'll have to implement this method:
 - `public void handle(MouseEvent event)`
- Here are some of `MouseEvent`'s methods:
 - `public double getX()`
 - `public double getY()`
 - `public MouseButton getButton()`

WindowEvent: inner class syntax

```
public class StartPresenter {  
    //... constructor and attributes left out  
    private void addEventHandlers() {  
        view.getScene().getWindow().setOnCloseRequest(new ConfirmCloseHandler());  
    }  
  
    private class ConfirmCloseHandler implements EventHandler<WindowEvent> {  
        @Override  
        public void handle(WindowEvent event) {  
            final Alert sure = new Alert(CONFIRMATION,  
                "Are you sure you want to exit?");  
            sure.setHeaderText("Are you sure?");  
            Optional<ButtonType> choice = sure.showAndWait();  
            if (choice.isPresent() &&  
                choice.get().getText().equalsIgnoreCase("CANCEL")) {  
                event.consume();  
            }  
        }  
    }  
}
```

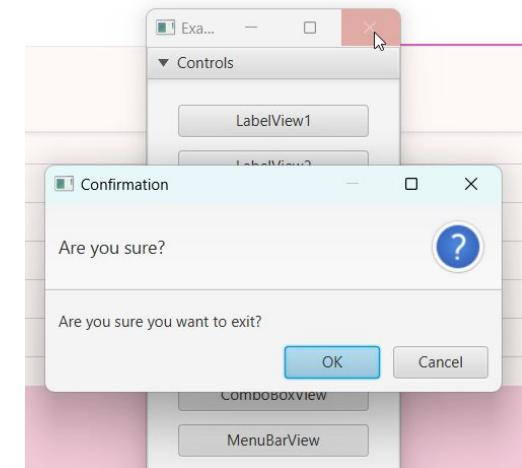
This is the Stage



WindowEvent: lambda syntax

```
public class StartPresenter {  
  
    private void addEventHandlers() {  
        view.getScene().getWindow().setOnCloseRequest(  
            event -> {  
                final Alert sure = new Alert(CONFIRMATION,  
                    "Are you sure you want to exit?");  
                sure.setHeaderText("Are you sure?");  
                Optional<ButtonType> choice = sure.showAndWait();  
                if (choice.isPresent() &&  
                    choice.get().getText().equalsIgnoreCase("CANCEL")) {  
                    event.consume();  
                } // end if  
            } // end lambda  
        ); // end call setOnCloseRequest  
    } // end addEventHandlers  
} // end class
```

Lambda's are most readable if they're short



WindowEvent

- An event handler for **KeyEvent** can be added to a Window (i.e. a Stage) or to a Tab. The method to use is:

→ `public void setOnCloseRequest(EventHandler<WindowEvent> handler)`

- The Window class supports more event handlers:

→ `public void setOnHidden(EventHandler<WindowEvent> handler)`

→ `public void setOnHiding(EventHandler<WindowEvent> handler)`

→ `public void setOnShown(EventHandler<WindowEvent> handler)`

→ `public void setOnShowing(EventHandler<WindowEvent> handler)`

- Use an **inner class or a lambda** that implements `EventHandler<WindowEvent>`

- You'll have to implement this method:

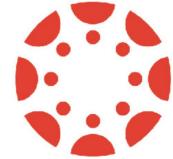
→ `public void handle(WindowEvent event)`

Assignment



Complete these assignments:

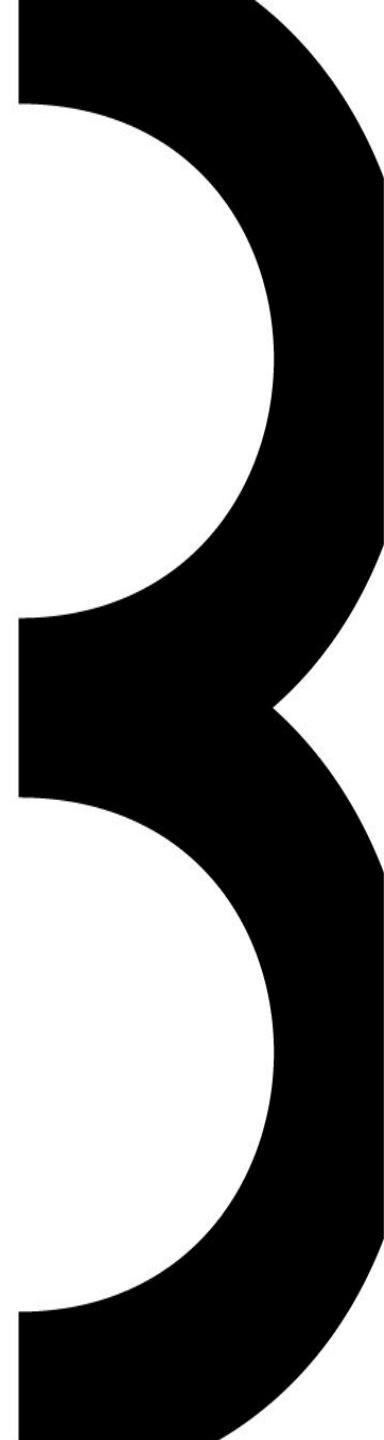
- Java 4.02: Time
- Java 4.03: Tiles



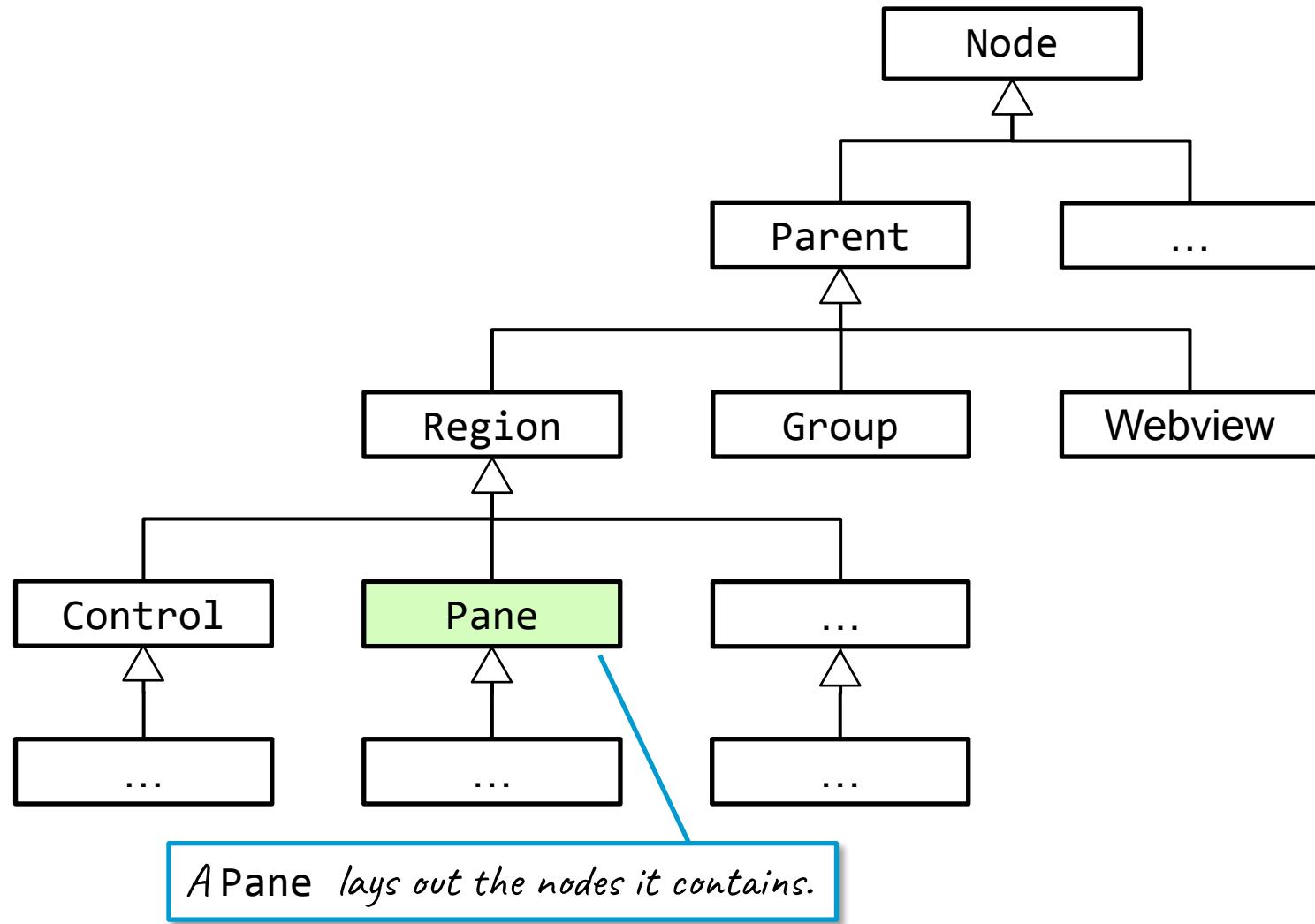
- You can start from the starter project
- Carefully read all code that is given to you



Layout



Pane within the class hierarchy

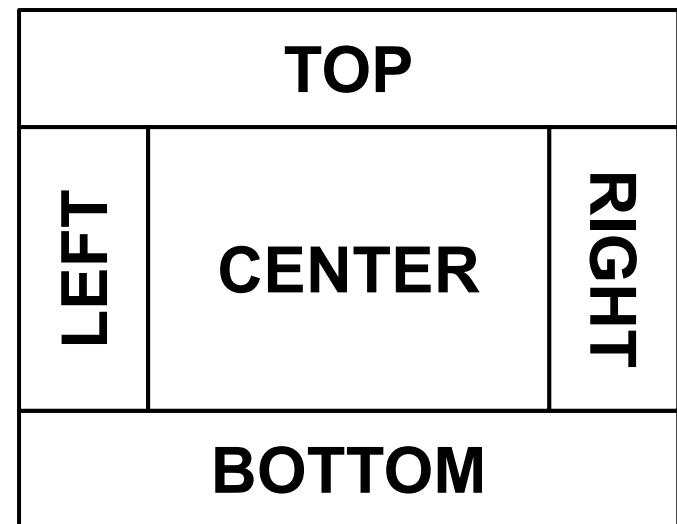


Pane: layout of Nodes

- We'll demonstrate:
 - BorderPane
 - FlowPane
 - GridPane
- For additional panes, check the documentation:
 - http://docs.oracle.com/javafx/2/layout/builtin_layouts.htm
 - <https://openjfx.io/javadoc/11/javafx.graphics/javafx.scene/layout/package-summary.html>

BorderPane

- A BorderPane has 5 regions
 - The Control in the central region scales horizontally and vertically up to its maximum width and height
 - Top and Bottom:
 - Horizontally: scales to maxWidth
 - Vertically: fixed, set to prefWidth
 - Left and Right: the opposite!
- Use setters to add nodes to a region
- static method `setMargin` sets the space around the nodes
- static method `setAlignment` aligns a node within its region



Common use: a menu in the top region and a status bar in the bottom region.

BorderPane

```
public class BorderPaneView extends BorderPane {  
    private MenuItem exit;  
    private TextArea area;  
    private Label status;  
    public BorderPaneView() {  
        initialiseNodes(); layoutNodes();  
    }  
    private void initialiseNodes() {  
        exit = new MenuItem("Exit");  
        area = new TextArea();  
        status = new Label("Waiting for text.");  
        status.setStyle("-fx-background-color: orange");  
    }  
    private void layoutNodes() {  
        Menu bestandMenu = new Menu("File", null, exit);  
        MenuBar menuBar = new MenuBar(bestandMenu);  
        setTop(menuBar);  
        setCenter(area);  
        setBottom(status);  
        status.setMaxWidth(Double.MAX_VALUE);  
    }  
}
```



The status bar and the menu only scale horizontally. The text area scales in both dimensions

Setting the background using CSS

To have a control scale infinitely, set its maxWidth/Height to Double.MAX_VALUE

VBox



```
public class VBoxView1 extends VBox {  
    private Label label;  
    private Button button;  
    private ImageView imageView;  
  
    public VBoxView1() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        label = new Label("One");  
        button = new Button("Click");  
        imageView = new ImageView("/angrybird.png");  
    }  
  
    private void layoutNodes() {  
        setSpacing(20);  
        setPadding(new Insets(20));  
        getChildren().addAll(label, button, imageView);  
    }  
}
```

- `VBox` lays out the nodes vertically, in a single column
- `setSpacing()` sets the space between the nodes
- `setAlignment()` specifies how nodes should be aligned
- **Adding nodes:**

`getChildren().addAll(...)` or
`getChildren().add(...)`



VBox

```
public class VBoxView2 extends VBox {  
    private Label label;  
    private Button button;  
    private ImageView imageView;  
  
    public VBoxView2() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        label = new Label("One");  
        button = new Button("Click");  
        imageView = new ImageView("/angrybird.png");  
    }  
  
    private void layoutNodes() {  
        button.setMaxWidth(Double.MAX_VALUE);  
        setSpacing(20);  
        setPadding(new Insets(20));  
        getChildren().addAll(label, button, imageView);  
    }  
}
```

The size of the nodes:

- Nodes scale up to their maxWidth
- By default, node heights are fixed and equal to their preferredHeight



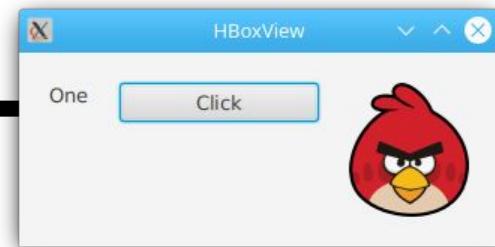
VBox

```
public class VBoxView3 extends VBox {  
    private Label label;  
    private Button button;  
    private ImageView imageView;  
  
    public VBoxView3() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        label = new Label("One");  
        button = new Button("Click");  
        imageView = new ImageView("/angrybird.png");  
    }  
  
    private void layoutNodes() {  
        button.setMaxWidth(Double.MAX_VALUE);  
        button.setMaxHeight(100);  
        VBox.setVgrow(button, Priority.ALWAYS);  
        setSpacing(20);  
        setPadding(new Insets(20));  
        getChildren().addAll(label, button, imageView);  
    }  
}
```

static method `setVgrow()` with value `Priority.ALWAYS` allows a node to grow to occupy available vertical space



HBox



```
public class HBoxView extends HBox {  
    private Label label;  
    private Button button;  
    private ImageView imageView;
```

```
public HBoxView() {  
    initialiseNodes();  
    layoutNodes();  
}
```

```
private void initialiseNodes() {  
    label = new Label("One");  
    button = new Button("Click");  
    imageView = new ImageView("angrybird.png");  
}
```

- HBox children are horizontal, in a single row
- Children scale vertically up to maxHeight
- By default, children have a fixed width equal to their preferredWidth
- Enable horizontal scaling by calling the static method setHgrow() with Priority.ALWAYS

```
private void layoutNodes() {  
    button.setMaxHeight(Double.MAX_VALUE);  
    HBox.setHgrow(button, Priority.ALWAYS);  
    setSpacing(20);  
    setPadding(new Insets(20));  
    getChildren().addAll(label, button, imageView);  
}
```

FlowPane

- FlowPane lays out its children in rows or columns, very much like HBox and VBox
 - When there is not enough room in a row/column, FlowPane will wrap to the next row/column.
 - FlowPane will however not resize its children to occupy remaining space

GridPane



```
public class GridPaneView1 extends GridPane {  
    private Label label;  
    private Button button;  
    private ImageView imageView;  
  
    public GridPaneView1() {  
        initialiseNodes();  
        layoutNodes();  
    }  
  
    private void initialiseNodes() {  
        label = new Label("One");  
        button = new Button("Click");  
        imageView = new ImageView("/angrybird.png");  
    }  
  
    private void layoutNodes() {  
        add(label, 0, 0);  
        add(imageView, 1, 0);  
        add(button, 0, 1);  
        setGridLinesVisible(true);  
    }  
}
```

- lays out nodes in a grid
- add() specifies in which cell of the grid the node is to be added
- Use setGridLinesVisible to show the borders of the cells

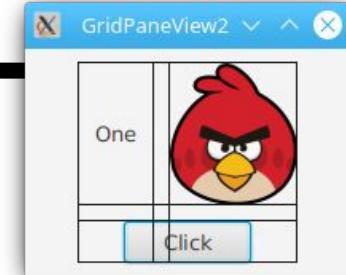
Column

Row

GridPane

```
// ...
```

```
private void layoutNodes() {  
    label.setPadding(new Insets(10));  
    button.setPrefSize(80, 20);  
    add(label, 0, 0);  
    add(imageView, 1, 0);  
    add(button, 0, 1, 2, 1);  
  
    setAlignment(Pos.CENTER);  
    setHgap(10);  
    setVgap(10);  
  
    GridPane.setAlignment(button, HPos.CENTER);  
  
    setGridLinesVisible(true);  
}
```



GridPane uses the preferred size

button spans 2 columns and 1 row

Use setAlignment() to specify the grid's alignment within its parent

Use setHgap() and setVgap() to set the space between cells

setH/Valignment() can be used to align nodes within the grid

```
// ...
```

GridPane: constraints

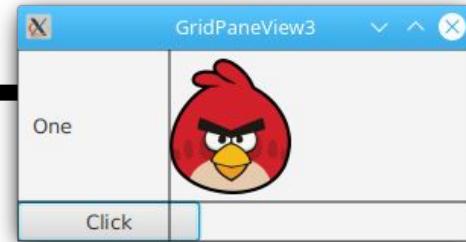
```
// ...
private void layoutNodes() {
    label.setPadding(new Insets(10));
    button.setPrefSize(120, 20);
    add(label, 0, 0);
    add(imageView, 1, 0);
    add(button, 0, 1, 2, 1);
```

ColumnConstraints column1 = new ColumnConstraints(100);
ColumnConstraints column2 = new ColumnConstraints(200);
getColumnConstraints().addAll(column1, column2);

RowConstraints rowConstraints = new RowConstraints(100);
getRowConstraints().addAll(rowConstraints);

setGridLinesVisible(true);

}

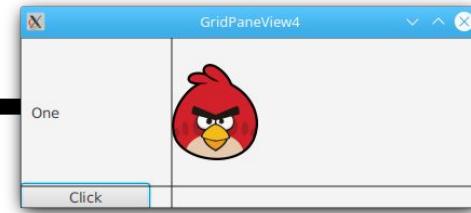


First column 100 pixels wide,
Second one 200 pixels wide.
First row 100 pixels high

- Column/RowConstraints set the properties of rows and columns
- Add to getColumn/RowConstraints() in the right order using the method addAll()

GridPane: constraints

```
private void layoutNodes() {  
    label.setPadding(new Insets(10));  
    button.setPreferredSize(120, 20);  
    add(label, 0, 0);  
    add(imageView, 1, 0);  
    add(button, 0, 1, 2, 1);
```



Roughly 1/3 and 2/3 wide

```
ColumnConstraints column1 = new ColumnConstraints();  
column1.setPercentWidth(33.0);  
ColumnConstraints column2 = new ColumnConstraints();  
column2.setPercentWidth(67.0);  
getColumnConstraints().addAll(column1, column2);
```

```
RowConstraints row1 = new RowConstraints();  
row1.setVgrow(Priority.ALWAYS);  
RowConstraints row2 = new RowConstraints(20.0);  
getRowConstraints().addAll(row1, row2);
```

The bottom row is 20 pixels high, all remaining space is left to the top row

```
}  
setGridLinesVisible(true);
```

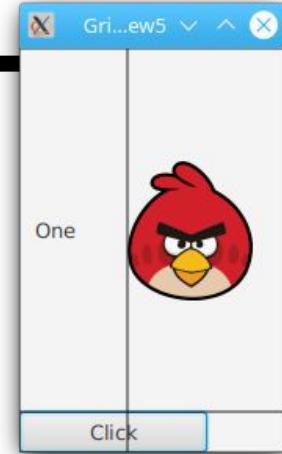
- It's possible to use a percentage
- Vgrow and Hgrow can be used as well
- In both cases, the GridPane will occupy all available space

GridPane: constraints

```
private void layoutNodes() {  
    label.setPadding(new Insets(10));  
    button.setPrefSize(120, 20);  
    add(label, 0, 0);  
    add(imageView, 1, 0);  
    add(button, 0, 1, 2, 1);  
  
    GridPane.setConstraints(label,  
        HPos.LEFT, Priority.ALWAYS,  
        0, 0, 1, 1,  
        VPos.CENTER, Priority.ALWAYS);  
    GridPane.setConstraints(imageView,  
        HPos.LEFT, Priority.ALWAYS,  
        1, 0, 1, 1,  
        VPos.CENTER, Priority.ALWAYS);  
    GridPane.setConstraints(button,  
        HPos.LEFT, Priority.ALWAYS,  
        0, 1, 2, 1,  
        VPos.CENTER, Priority.NEVER);  
  
    setGridLinesVisible(true);  
}
```

The same result can be obtained by specifying properties at the cell level

The **label** and **imageView** cells will grow both vertically and horizontally



The **button**'s cell will not grow vertically

Assignment



Complete these assignments:

- Java 4.04: City Hall (VBox)
- Java 4.05: Calculator (GridPane)



- Start from the starter project
- Carefully read the provided code

**Model - View -
Presenter**

Model - View - Presenter

• Model – View – Presenter:

- Main class is standard (boilerplate)
- Update your MVP template project



- **NO** business-logic in Main class



- Main class is in the 'appname',
NOT in 'model' or 'view'

Model View Presenter (MVP) v.1

```
Package applicationname
Main.java
package applicationname;
import ...;

public class Main extends Application {
    @Override
    public void start(Stage primaryStage) {
        ApplicationNameModel model = new ApplicationNameModel();
        ApplicationNameView view = new ApplicationNameView();
        new ApplicationNamePresenter(model, view);
        primaryStage.setScene(new Scene(view));
        primaryStage.show();
    }

    public static void main(String[] args) {
        Application.launch(args);
    }
}

Package applicationname.model
ApplicationNameModel.java
package applicationname.model;
import ...;

public class ApplicationNameModel {
    // private attributes
    public ApplicationNameModel() {
    } // Constructor

    // methods with business logic
    // needed getters and setters
}
```

```
Package applicationname.view
ApplicationNameView.java
package applicationname.view;
import ...;

public class ApplicationNameView extends /* layout type */ {
    // private Node attributes (controls)
    public ApplicationNameView () {
        initialiseNodes();
        layoutNodes();
    }

    private void initialiseNodes() {
        // create and configure controls
        // button = new Button("...") 
        // label = new Label("...")
    }

    private void layoutNodes() {
        // add/set - methods
        // Insets, padding, alignment ...
    }

    // package-private Getters
    // for controls used by Presenter
}
```

```
Package applicationname.presenter
ApplicationNamePresenter.java
package applicationname.presenter;
import ...;

public class ApplicationNamePresenter {
    private ApplicationNameModel model;
    private ApplicationNameView view;

    public ApplicationNamePresenter(
        ApplicationNameModel model,
        ApplicationNameView view) {
        this.model = model;
        this.view = view;
        addEventHandlers();
        updateView();
    }

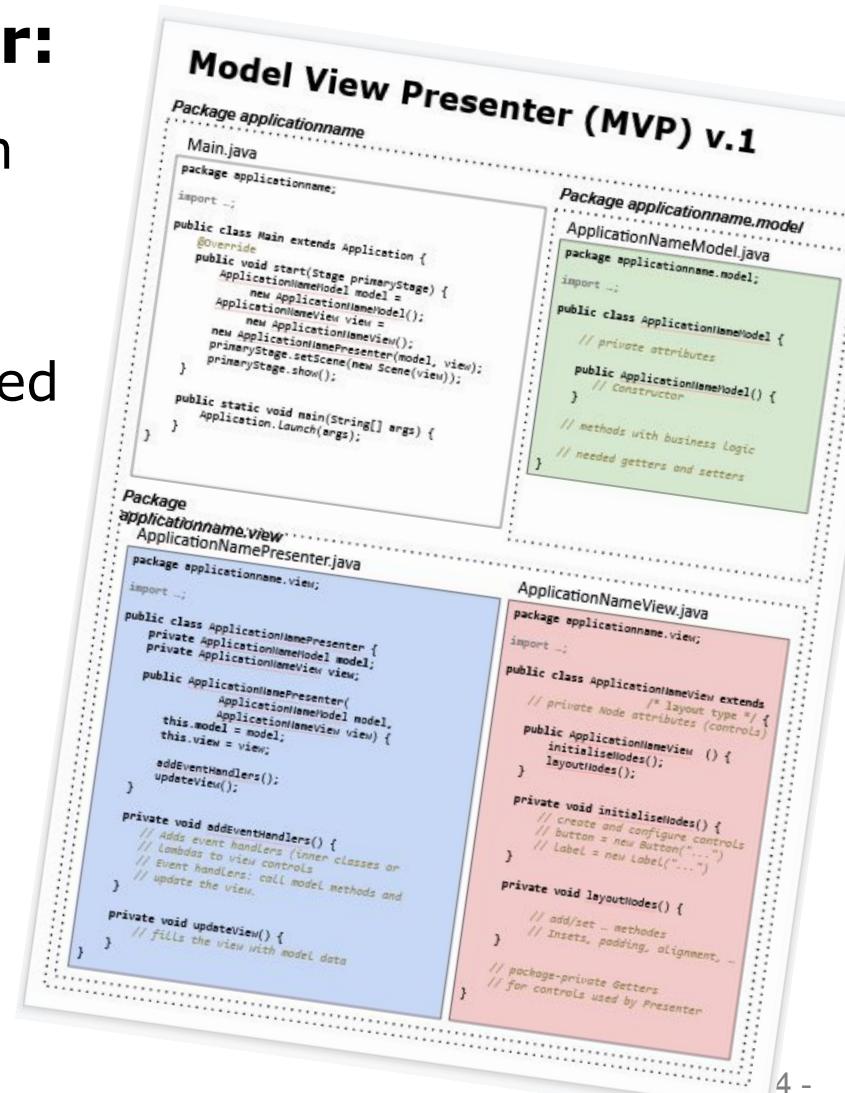
    private void addEventHandlers() {
        // Adds event handlers (inner classes or
        // Lambdas to view controls
        // Event handlers: call model methods and
        // update the view.
    }

    private void updateView() {
        // fills the view with model data
    }
}
```

Model - View - Presenter

• Model – View – Presenter:

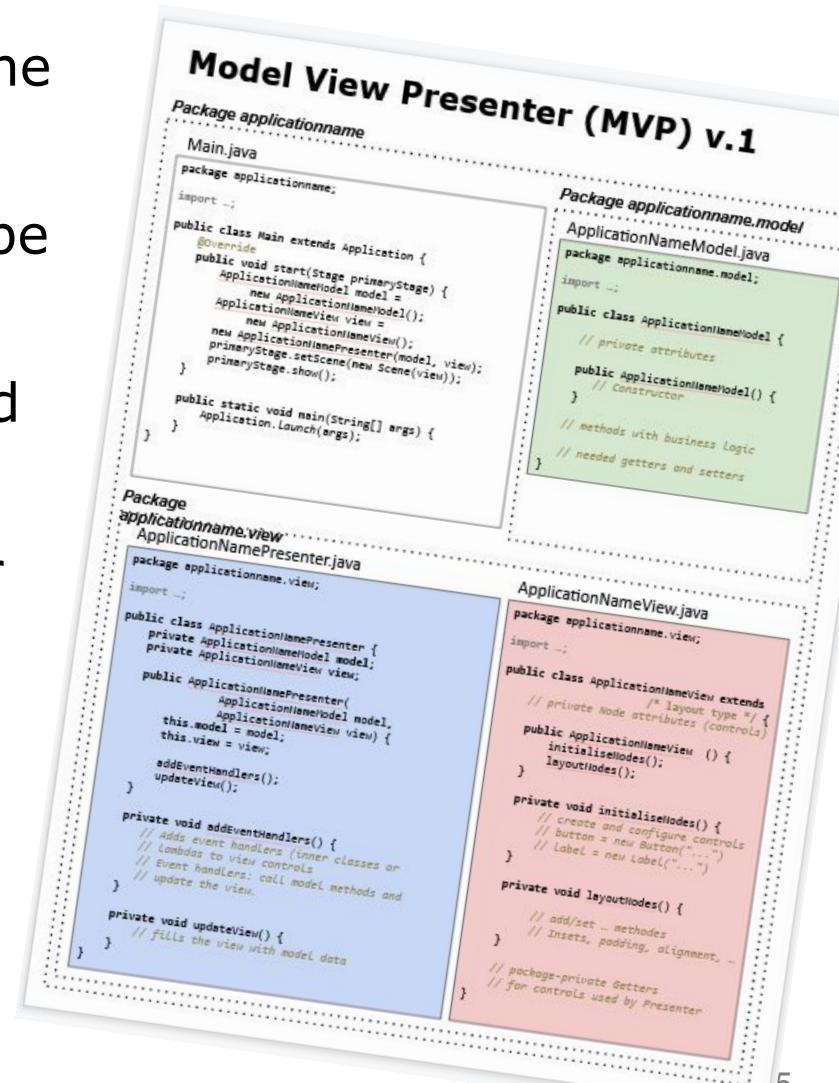
- **DON'T** call the JavaFX API from the model
- Variables are **private**. Use getters and setters where needed



Model - View - Presenter

• Model – View – Presenter:

- All UI-related code belongs in the 'view' package
- Getters in View classes should be package-private, **NOT** public
- Setters in View classes can (and should) be avoided
- Event-handling in the Presenter
 - Delegate detailed handling of graphical components to the view





Summary

- Controls
 - Label, ImageView, Button, CheckBox, TextField, ComboBox, Menu's
- Events
 - ActionEvent – MouseEvent – KeyEvent – WindowEvent
 - Lambdas
- Layout
 - BorderPane – HBox – VBox – GridPane
- Model - View - Presenter



Multi Window MVP applications

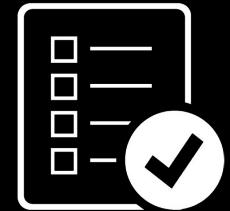
Graphics

Charts

Programming 2

Agenda

1. MVP multi window package structure
2. Start Scene on current Stage
3. Start Scene on new Stage()
4. Alert
5. MVP multi window summary
6. Graphics
7. Background, Node Effects
8. Charts
9. Exceptions and MVP



Multi Window MVP Applications

MVP Multi window package structure



[Example code](#)

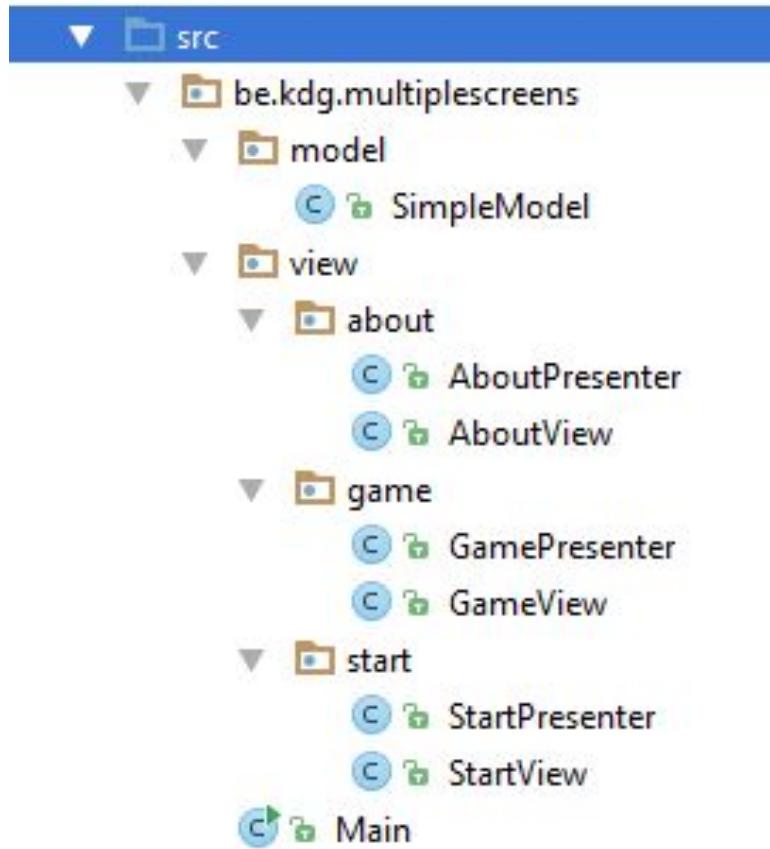
Example: Game of Goose

- The application consists of
 - **Start scene:** displays a start button in the primaryStage. Clicking the button replaces the screen with the game screen (*on the same stage!*)
 - **Game scene:** displays the game board. Closing the window shows an alert to confirm you want to quit.
 - **About scene:** the help menu opens the about scene. It is shown on a separate stage.

- just shows the scenes, no real game



Every scene has its View and Presenter



- A package for every scene
- Contains a View and a Presenter
- These are simple views.
Complex views could consist of multiple classes

Multi Window MVP Applications

**Start Scene on
current stage**

Scene A -> Scene B on current Stage

- Swap scenes on a Stage
- Game of Goose: Start Scene -> Game Scene
 - Swap StartView with GameView (same Stage)
 - By StartPresenter (A presenter)
 - Creates GameView (B View)
 - Changes Scene root to GameView
 - Calls `sizeToScene` to size Stage to current View in Scene
 - Creates GamePresenter (B Presenter) and passes model and GameView to it

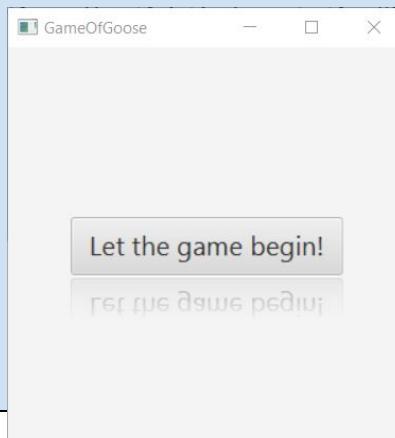
Start Scene -> Game Scene

StartPresenter.java

```
//...
private void addEventHandlers() {
    view.getBtnNextScreen().setOnAction(event -> setGameView()) ;
}

private void setGameView() {
    GameView gameView = new GameView();
    view.setScene().setRoot(gameView);
    gameView.setScene().getWindow().sizeToScene();
    GamePresenter gamePresenter =
        new GamePresenter(model, gameView);
}
//...
```

The application model is passed to the GamePresenter



Multi Window MVP Applications

**Start Scene on
new Stage()**

Scene A -> Scene B on new Stage()

- Create a new Stage to show a new Scene
 - allows to return to the previous scene
- Game of Goose: Game Scene (A) -> About Scene (B)
- The about menu shows a new about modal window
 - on top of the existing (parent) window, blocking access to it
- By GamePresenter (A Presenter)
 - Creates AboutView (B view)
 - Creates Stage B
 - Display Stage B over current Stage A (initOwner,initModality)
 - Creates Scene B and puts AboutView on Scene
 - Creates AboutPresenter (B presenter) and passes model and AboutView to it
 - Shows Scene B on Stage

Game Scene -> About Scene

GamePresenter.java

```
//...
private void addEventHandlers() {
    view.getMiAbout().setOnAction(event -> { showAbout(); });
}

private void showAbout() {
    AboutView aboutView = new AboutView();
    Stage aboutStage = new Stage();
    aboutStage.initOwner(view.getScene().getWindow());
    aboutStage.initModality(Modality.APPLICATION_MODAL);
    aboutStage.setScene(new Scene(aboutView));
    aboutStage.setX(view.getScene().getWindow().getX() + 100);
    aboutStage.setY(view.getScene().getWindow().getY() + 100);
    AboutPresenter aboutPresenter =
        new AboutPresenter(model, aboutView);
    aboutStage.showAndWait();
}
// ...
```

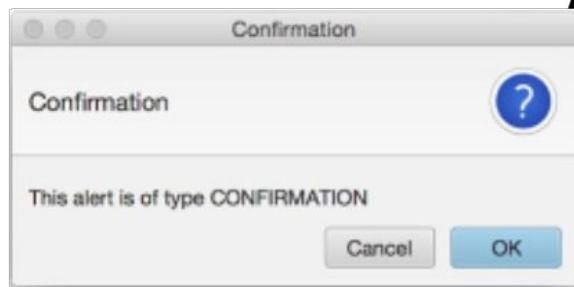
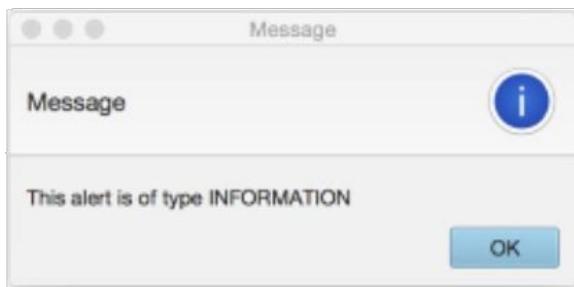
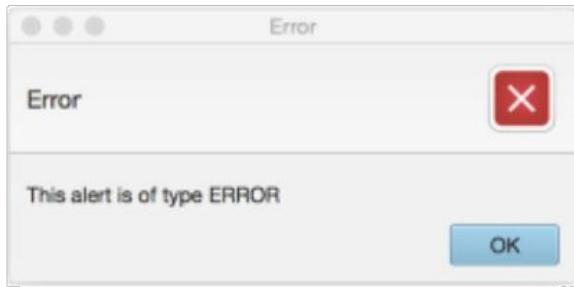
A MODAL window blocks user interface access to the owning window

Optional: the default position is close to this

window blocks (no window event handler)



Alerts



Show Alert

- Show a popup
 - allows to return to the previous scene
 - Game of Goose: Alert before closing the application
- When the user closes the Stage, show an alert
- Requires a WindowEventHandler
- By StartPresenter: First Presenter to use the Stage
 - adds a WindowEventHandler to the stage
 - the code in the EventHandler shows an alert when the window is closed and handles the user response

Show Alert upon closing window

StartPresenter.java

```
//...
public void addWindowEventHandlers() {
    view.getScene().getWindow().setOnCloseRequest(event -> {
        closeApplication(event);
    });
}

private void closeApplication(WindowEvent event) {
    Alert alert = new Alert(Alert.AlertType.WARNING);
    alert.setHeaderText("You are about to quit the game!");
    alert.setContentText("Do you really want to leave?");
    alert.setTitle("Hark Hark!");
    alert.getButtonTypes().clear();
    ButtonType no = new ButtonType("NO");
    ButtonType yes = newButtonType("YES");
    alert.getButtonTypes().addAll(no, yes);
    alert.showAndWait();
    if (alert.getResult() == null
        || alert.getResult().equals(no))
        event.consume();
}
//...
```



Show Alert upon closing window

StartPresenter.java

```
//...
private void addEventHandlers() {
    view.getScene().getWindow().setOnCloseRequest(event ->
closeApplication(event));
    // other event handlers
}

private void closeApplication(WindowEvent event) {
    Alert alert = new Alert(Alert.AlertType.WARNING);
    alert.setHeaderText("You are about to quit the game!");
    alert.setContentText("Do you really want to leave?");
    alert.setTitle("Hark Hark!");
    alert.getButtonTypes().clear();
    ButtonType no = new ButtonType("NO");
    ButtonType yes = newButtonType("YES");
    alert.getButtonTypes().addAll(no, yes);
    alert.showAndWait();
    if (alert.getResult() == null
        || alert.getResult().equals(no))
        event.consume();
}
//...
```



MVP multiwindow summary

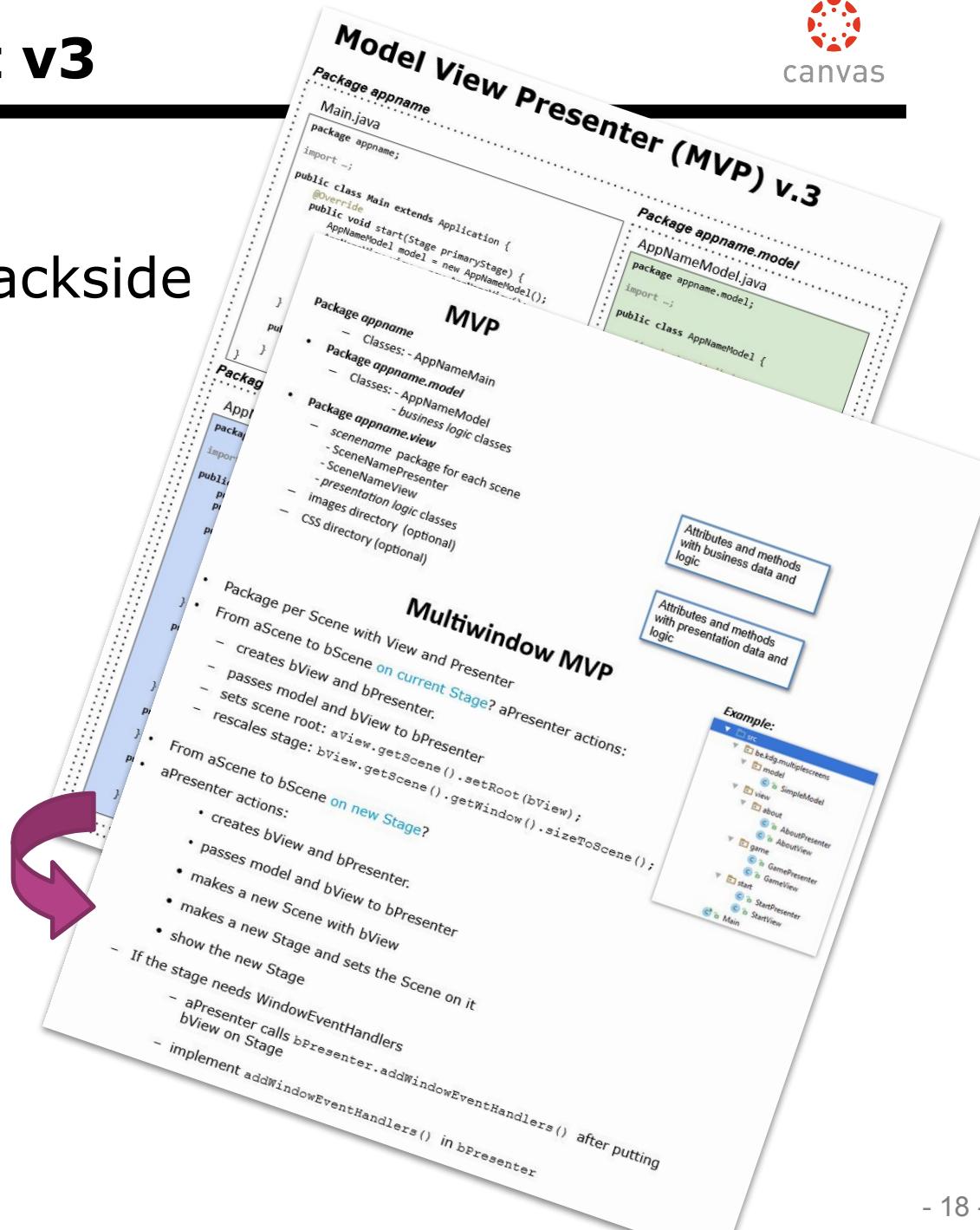
MVP multi window guidelines



- From aScene to bScene **on new Stage?**
- aPresenter actions:
 - creates bView and bPresenter.
 - passes model and bView to bPresenter
 - makes a new Scene with bView
 - makes a new Stage and sets the Scene on it
 - show the new Stage
- If the stage needs WindowEventHandlers add them to the Presenter's addEventHandlers method

MVP Cheat Sheet v3

- Guidelines on backside
- Final version



Exercises



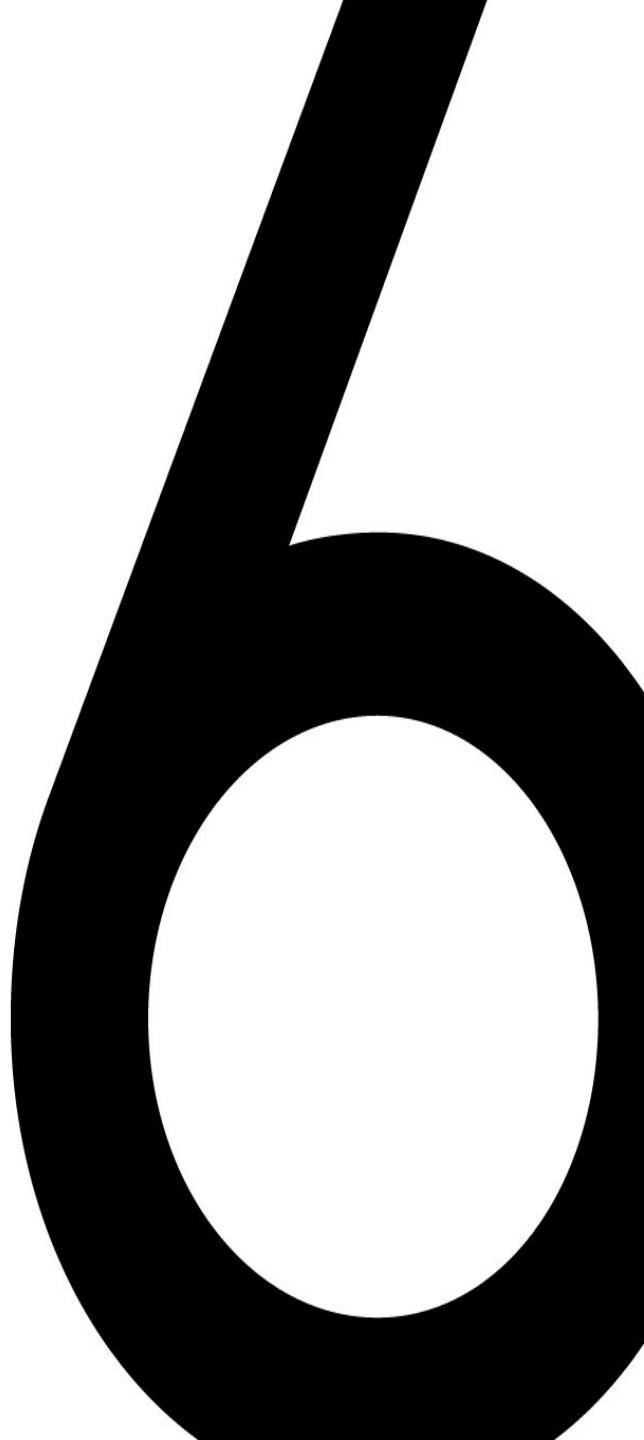
- Complete exercise Java 5.01

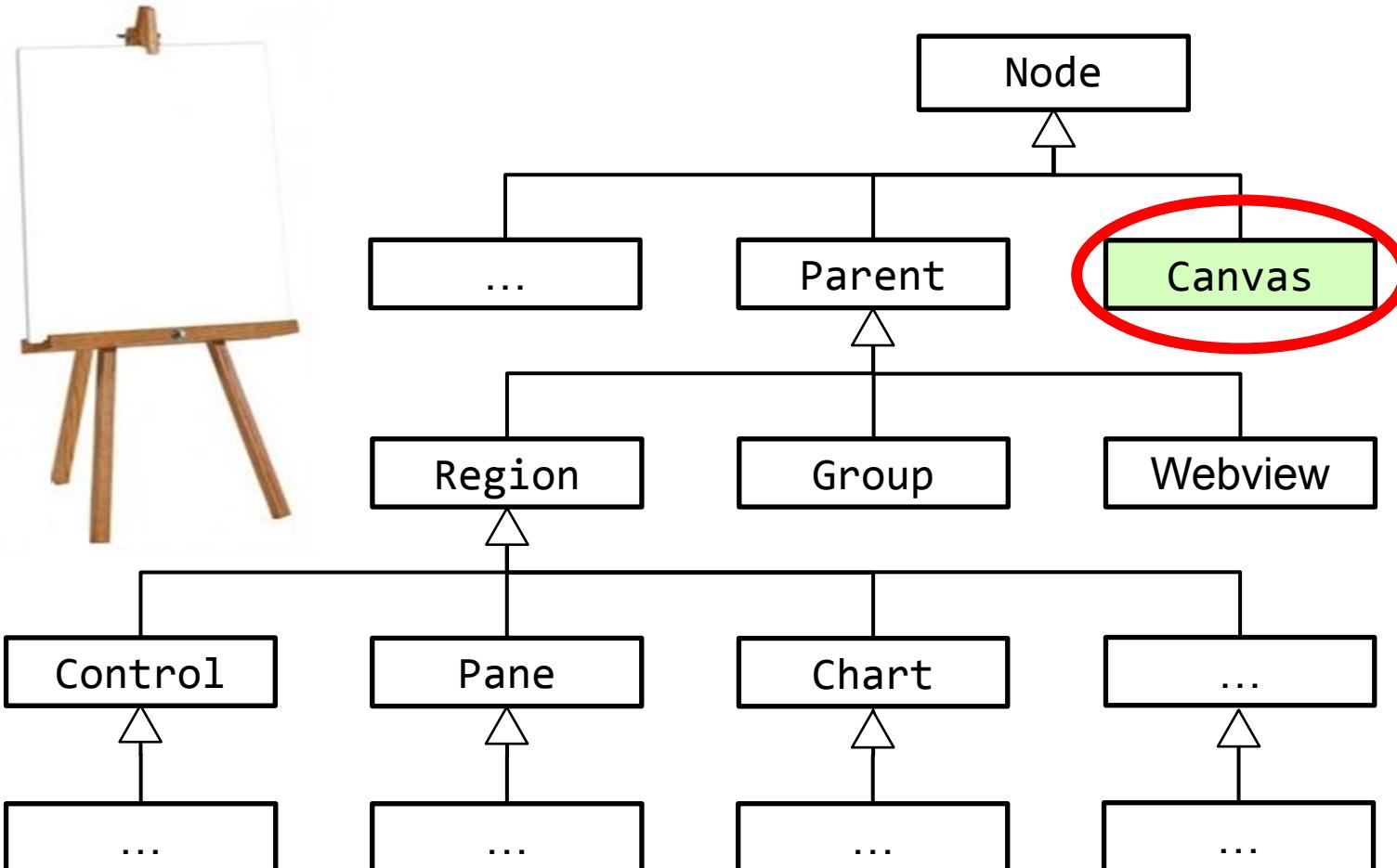
Graphics



Examples

Example

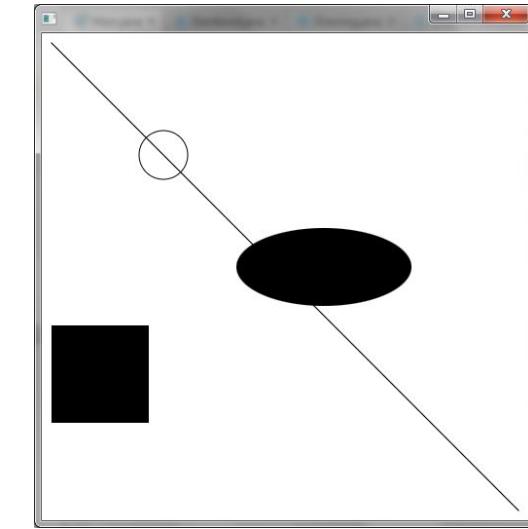




Draw on a canvas using GraphicsContext



```
public class CanvasView extends BorderPane {  
    private Canvas canvas;  
  
    public CanvasView() {  
        initialiseNodes();  
        layoutNodes();  
        draw();  
    }  
    private void initialiseNodes() {  
        canvas = new Canvas(500, 500);  
    }  
    private void layoutNodes() {  
        setCenter(canvas);  
    }  
    void draw() {  
        GraphicsContext gc = canvas.getGraphicsContext2D();  
        gc.strokeLine(10, 10, 490, 490);  
        gc.strokeOval(100, 100, 50, 50);  
        gc.fillOval(200, 200, 180, 80);  
        gc.fillRect(10, 300, 100, 100);  
    }  
}
```



Initial width, height

GraphicsContext contains all draw methods

Typically, the draw method's purpose is to draw the model (data) and is triggered from the Presenter → package access



Coordinates

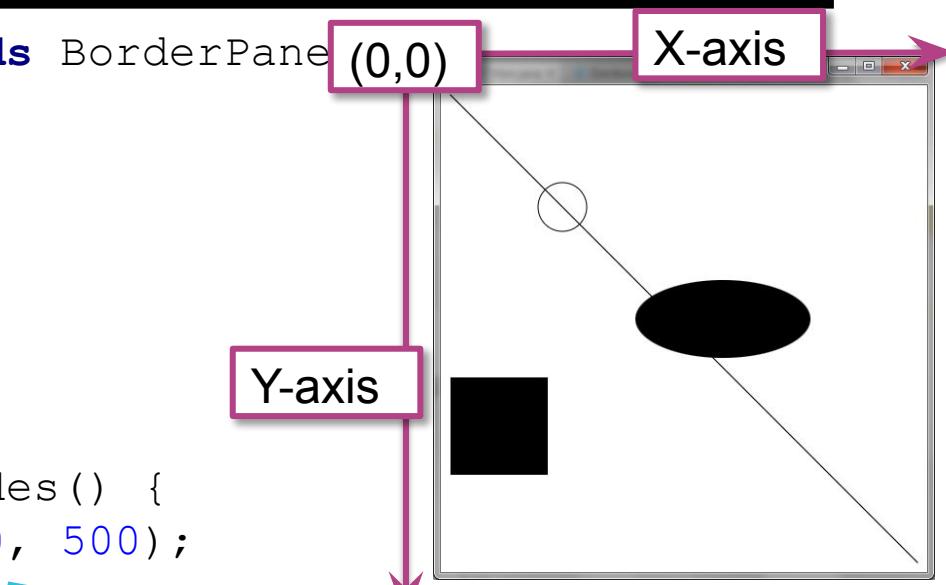
```
public class CanvasView extends BorderPane
    private Canvas canvas;

    public CanvasView() {
        initialiseNodes();
        layoutNodes();
        draw();
    }

    private void initialiseNodes() {
        canvas = new Canvas(500, 500);
    }

    private void layoutNodes() {
        setCenter(canvas);
    }

    void draw() {
        GraphicsContext gc = canvas.getGraphicsContext2D();
        gc.strokeLine(10, 10, 490, 490);
        gc.strokeOval(100, 100, 50, 50);
        gc.fillOval(200, 200, 180, 80);
        gc.fillRect(10, 300, 100, 100);
    }
}
```



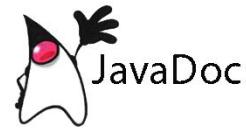
Y-axis

X-axis

Initial width, height

*GraphicsContext contains
all draw methods*

Draw on a canvas using GraphicsContext



- Draw on GraphicsContext:
 - stroke/fillLine: Line from (x1,y1) to (x2,y2)
 - stroke/fillRect: Open/filled rectangle. Left top at (x1, y1), width to right, height down.
 - stroke/fillOval: Open/filled ovals (and circles).
(x1, y1) = left top of surrounding box.
 - stroke/fillText: write on canvas.
 - drawImage: put picture (photo...) on canvas
 - clearRect: clears box
 - ...
- Set styles on GraphicsContext for all subsequent drawings
 - setFont: for Text
 - rotate: draw at an angle
 - scale: scales width and height
 - setLineWidth: stroke (pen) width
 - setLineDashes: set length of dashes (and gaps) of line
 - ...

Colors

- `setStroke (Paint p): line (pen) Paint`
 - `setFill (Paint p): fill Paint`
 - Paint can be
 - Color
 - ImagePattern
 - Lineair/RadialGradient: from/to color
 - Color
 - Predefinded:
 - `Color.BLACK, Color.BURLYWOOD, ...`
 - Mix your own colors:

```
Color c = Color.rgb(...);
```

```
Color c = Color.hsb(...);
```

```
Color c = new Color(...);
```

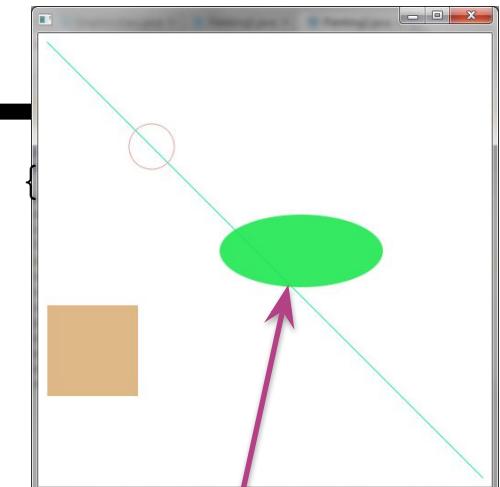
```
Color c = Color.web(...);
```
- Check javadoc and experiment!



RGB: *Red, Green, Blue* values
HSB: *Hue, Saturation, Brightness*
HSB is an alternate color specification often used in photography

Colors

```
public class ColorsView extends BorderPane {  
    //...  
  
    void draw() {  
        GraphicsContext gc  
            = canvas.getGraphicsContext2D();  
        gc.setStroke(Color.web("#2FEEA5"));  
        gc.strokeLine(10, 10, 490, 490);  
        gc.setStroke(Color.hsb(0.8, 0.3, 0.9));  
        gc.strokeOval(100, 100, 50, 50);  
        gc.setFill(Color.rgb(30, 230, 80, 0.9));  
        gc.fillOval(200, 200, 180, 80);  
        gc.setFill(Color.BURLYWOOD);  
        gc.fillRect(10, 300, 100, 100);  
    }  
}
```

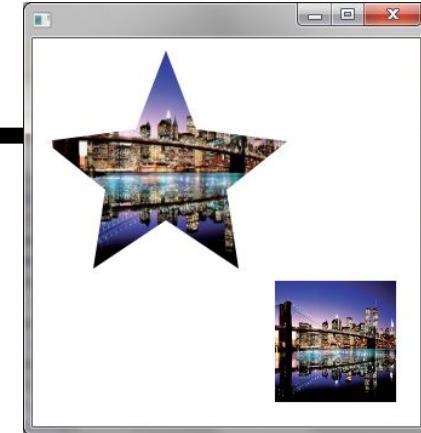


4th number is *transparency (alpha)*:
the lower, the more transparent. You
can see a faint line behind the oval..

FillPolygon - Path

1. setFill to ImagePattern
2. used for all subsequent fills

```
void draw() {  
    GraphicsContext gc = canvas.getGraphicsContext2D();  
    gc.setFill(new ImagePattern(new Image("newyork.jpg")));  
    double xpoints[] = {10, 85, 110, 135, 210, 160,  
                        170, 110, 50, 60};  
    double ypoints[] = {85, 75, 10, 75, 85, 125,  
                        190, 150, 190, 125};  
    gc.fillPolygon(xpoints, ypoints, xpoints.length);  
  
    gc.beginPath();  
    gc.moveTo(200, 200);  
    gc.lineTo(300, 200);  
    gc.lineTo(300, 300);  
    gc.lineTo(200, 300);  
    gc.closePath();  
    gc.fill();  
}
```



- Draw complex shapes with fill/drawPolygon: supply an array of x,y coordinates and JavaFx connects the dots!
- Alternatively draw a "path" with moveTo (lift pen) , lineTo (draw) ... methods

Exercises



- Complete exercises Java 5.02-5.04

Backgrounds Node effects

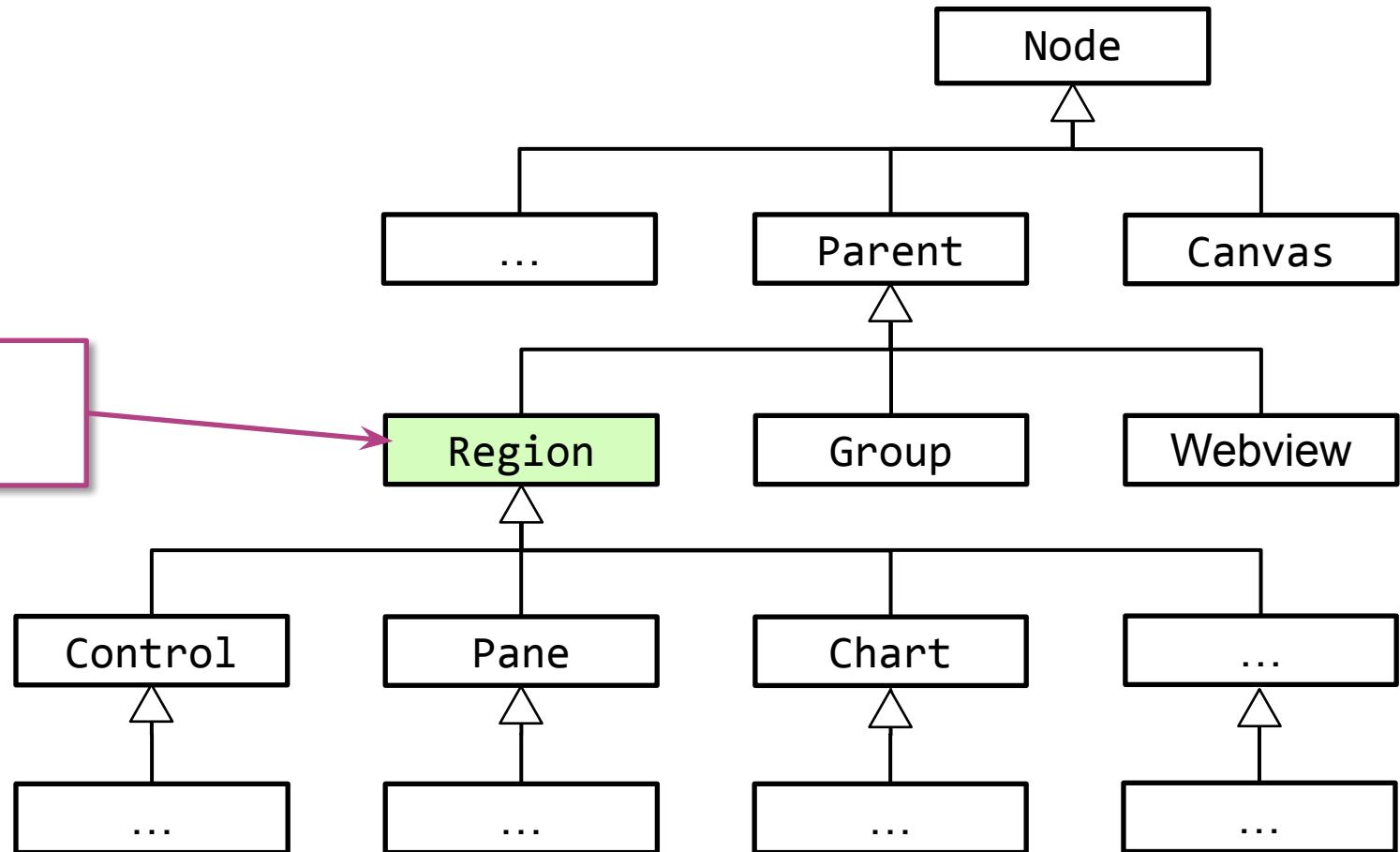
Background

- You can set a Background to any Region
- Using a BackgroundFill you can use a Paint as a Background



Background examples

A region can have a background



BackgroundFill: color

```
public class FillBackGroundView extends BorderPane {  
  
    private Canvas emptyCanvas;  
    private HBox hbox;  
    // constructor left out  
  
    private void initialiseNodes() {  
        // menubar code left out  
        emptyCanvas = new Canvas(480, 360);  
        hbox = new HBox(emptyCanvas);  
        hbox.setBackground(new Background(  
            new BackgroundFill(  
                Color.rgb(30, 230, 80, 0.9),  
                new CornerRadii(90),  
                new Insets(10)  
            )));  
    }  
  
    private void layoutNodes() {  
        setCenter(emptyCanvas);  
    }  
}
```



background covers entire HBox (center)

Paint (Color)

corner is part of a circle with radius 90 pixels. 0 results in a straight corner.

Border that is not filled. 0 indicates no border.

BackgroundImage

```
Image background = new Image("/images/goosegame.jpg");
pane = new HBox();
pane.setBackground(new Background(new BackgroundImage(
    background,
    BackgroundRepeat.NO_REPEAT,
    BackgroundRepeat.NO_REPEAT,
    BackgroundPosition.DEFAULT,
    new BackgroundSize(BackgroundSize.AUTO, BackgroundSize.AUTO, false,
    false, true, false)
)));
pane.setPrefWidth(background.getWidth());
pane.setPrefHeight(background.getHeight());
```

do not repeat horizontally or vertically if available room

Position relative to Region. **DEFAULT** = left top.
Can also be positioned at **BOTTOM**...

DEFAULT = same size as Region



Background is same size as pane (in CENTER), positioned at Center of BorderPane (under menu)

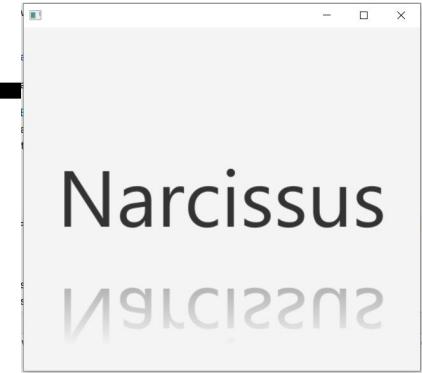
Node effects

```
public class EffectsPanel extends BorderPane {
    private Label text;

    public EffectsPanel() {
        initialiseNodes();
        layoutNodes();
    }

    private void initialiseNodes() {
        text = new Label("Narcissus");
        text.setFont(new Font(100));
        text.setEffect(new Reflection());
    }

    private void layoutNodes() {
        setCenter(text);
    }
}
```



- In class Node
- Graphical effects on JavaFx UI Node components
- Examples
 - DropShadow
 - Reflection
 - ...



Example code

Charts



[Example code](#)

Example



Elaboration

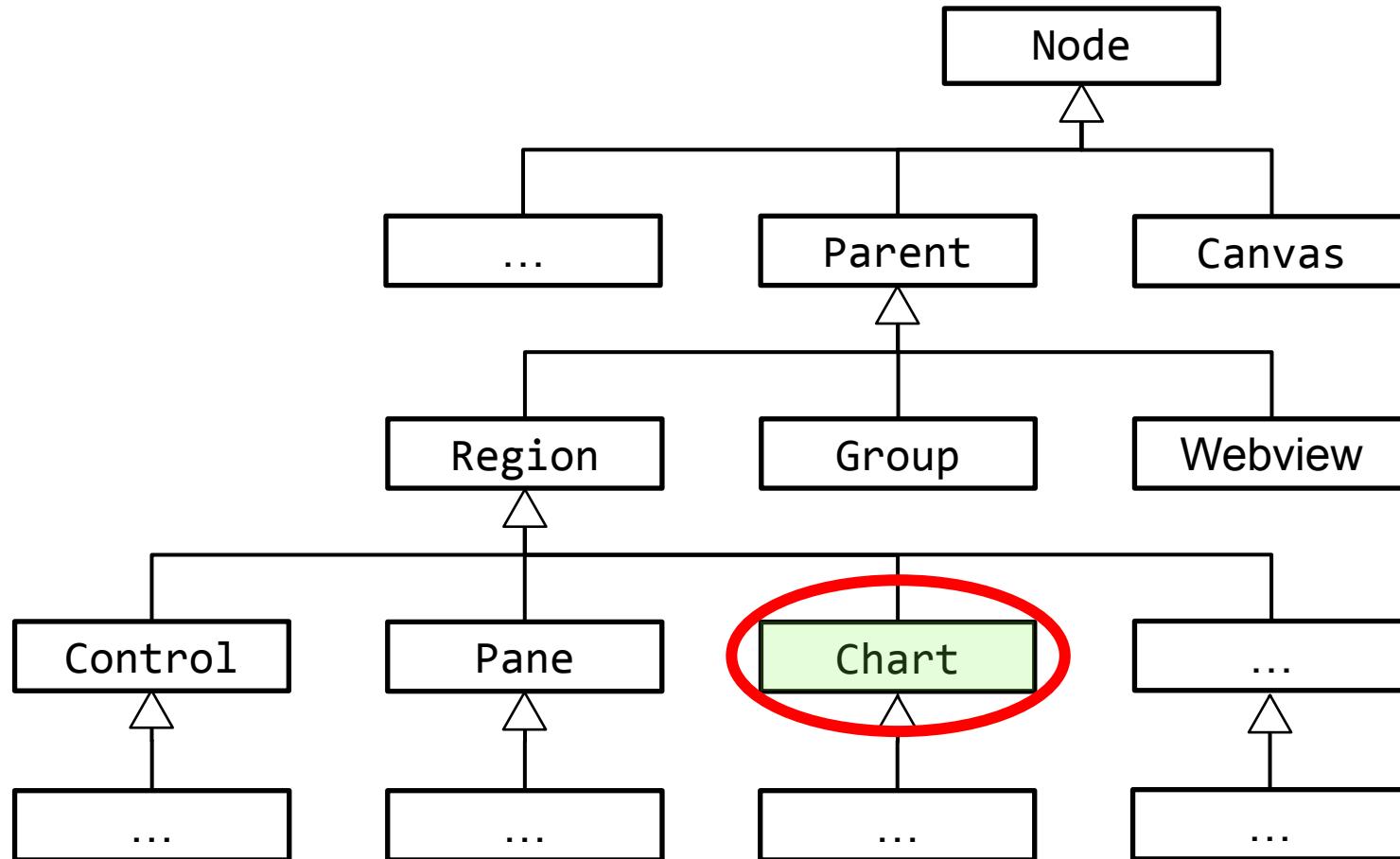
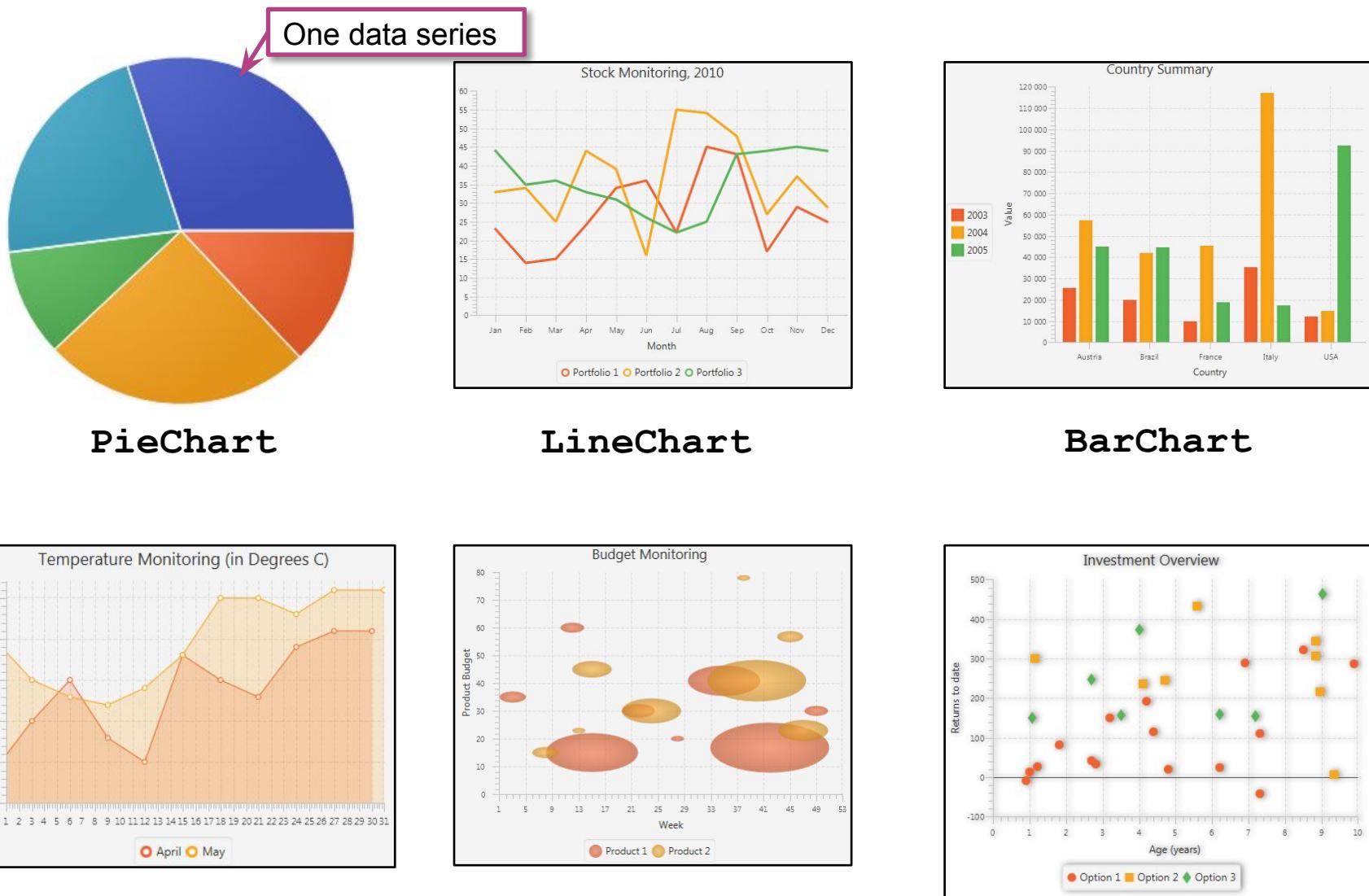


Chart Subclasses

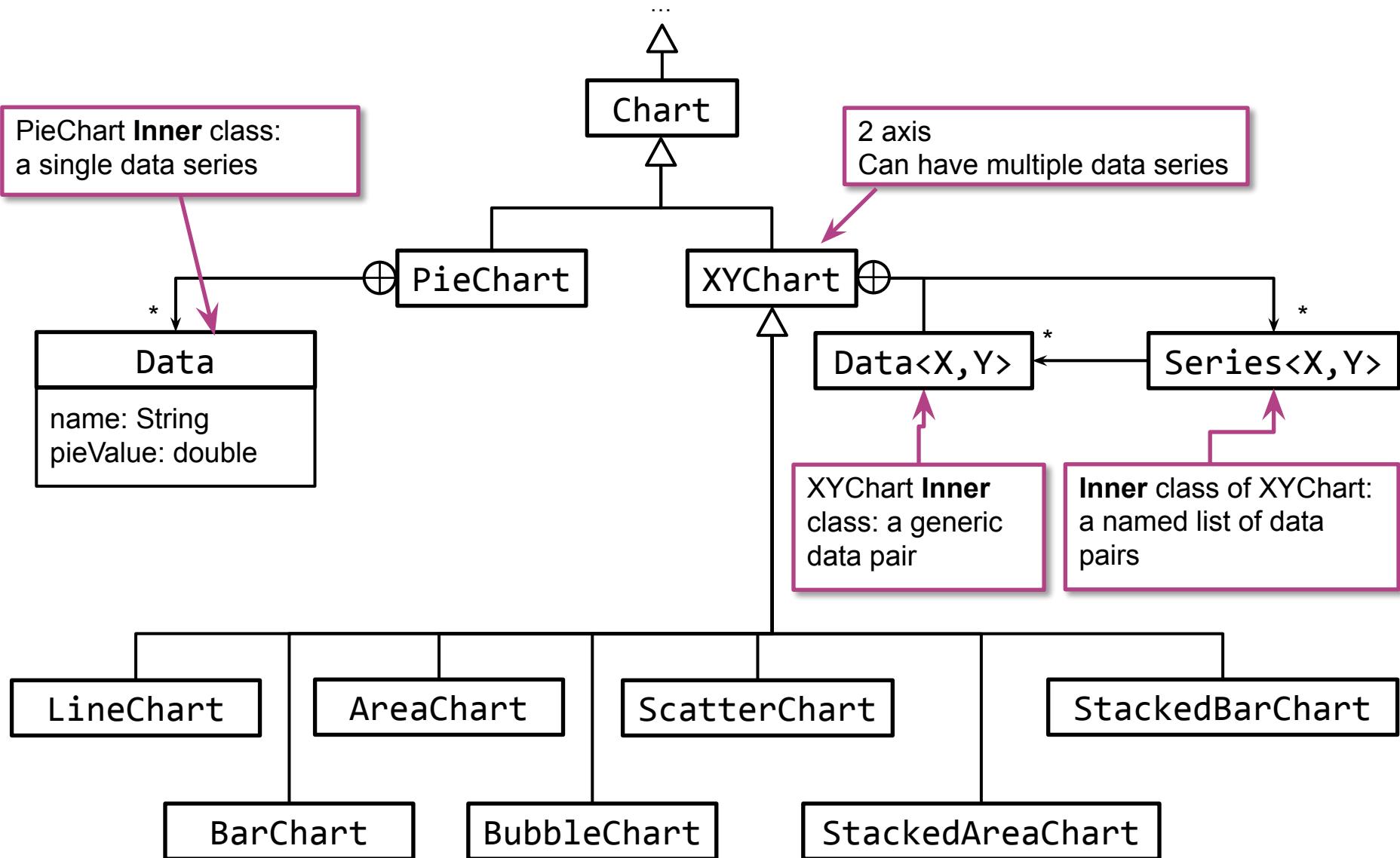


AreaChart

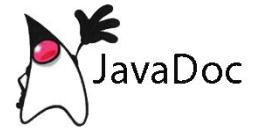
BubbleChart

ScatterChart

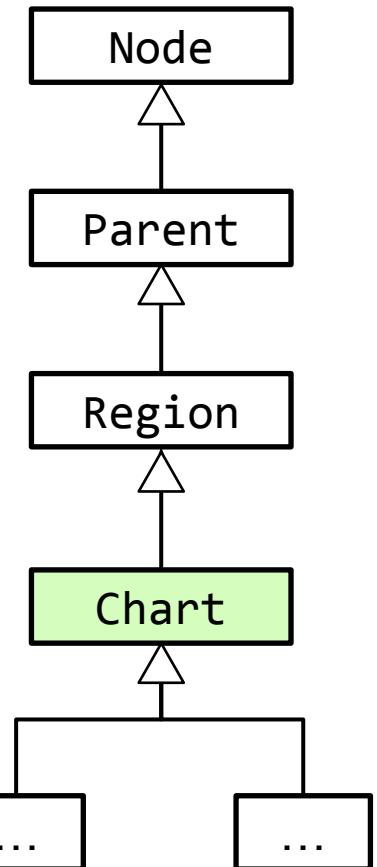
Chart subclasses



Charts: inherited methods



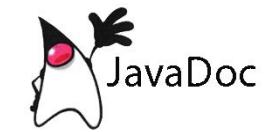
- set Pref/Min/Max Size/Width/Height
- setBorder
- setBackground
- setVisible
- setStyle → *Apply CSS*
- setDisabled
- setOn... → *EventHandlers*



→ *And many, many more!!*



Chart



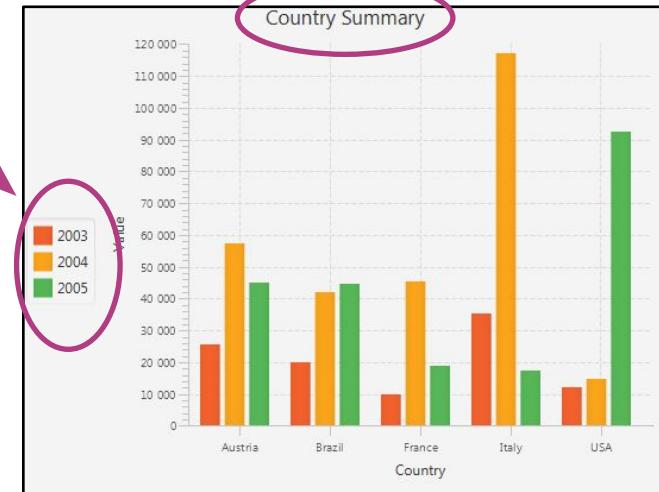
- Some methods...

Side.LEFT

`setTitle(String)`

`setTitleSide(Side)`

Side.TOP

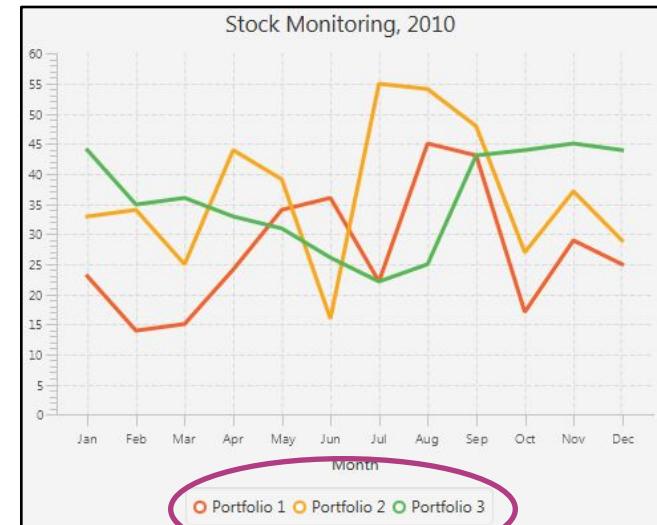


`setLegendSide(Side)`

`setLegendVisible(boolean)`

`setLegend(Node)`

Side.BOTTOM



PieChart

```
public class PieChartView extends BorderPane{
    private PieChart pie;

    public PieChartView() {
        initialiseNodes();
        layoutNodes();
    }

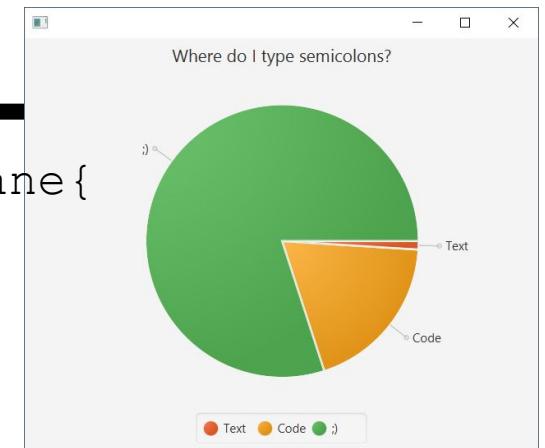
    private void initialiseNodes() {
        ObservableList<PieChart.Data> pieChartData =
            FXCollections.observableArrayList(
                new PieChart.Data("Text", 1),
                new PieChart.Data("Code", 19),
                new PieChart.Data(";", 80));
        this.pie = new PieChart(pieChartData);
    }

    private void layoutNodes() {
        this.pie.setTitle("Where do I type semicolons?");
        setCenter(this.pie);
    }
}
```

Single List of
PieChart.Data pairs

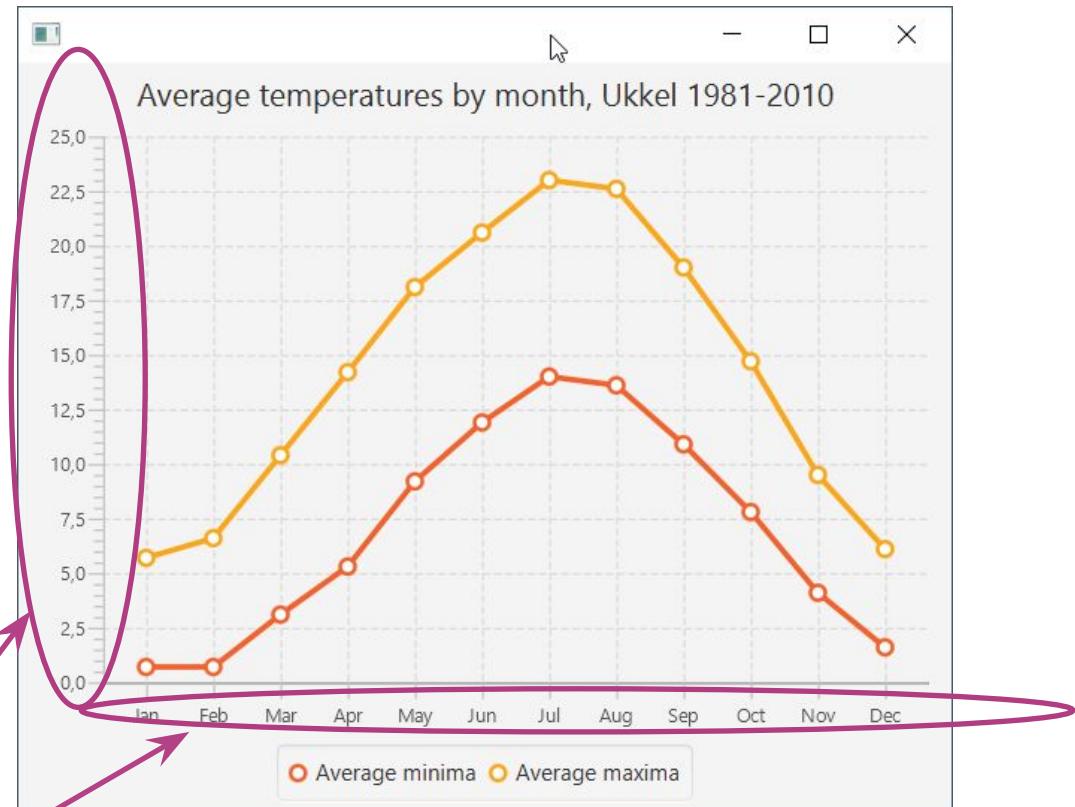
Each PieChart.Data object
has a name (String) and a value
(double)

You can add the data in the
PieChart constructor or
using the setData method.

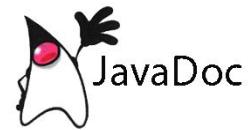


LineChart

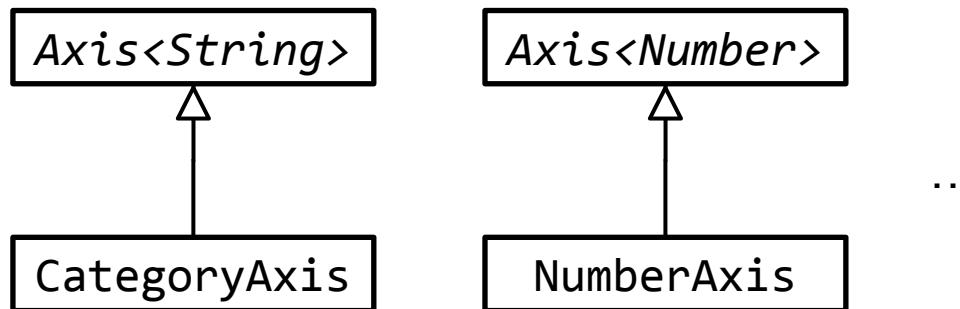
- each line is a Series of generic X,Y data pairs
- The chart can have multiple series
 - i. series 1: X,Y set of MINIMA data pairs
 - ii. series 2: X,Y set of MAXIMA data pairs
- Other XYCharts work similarly



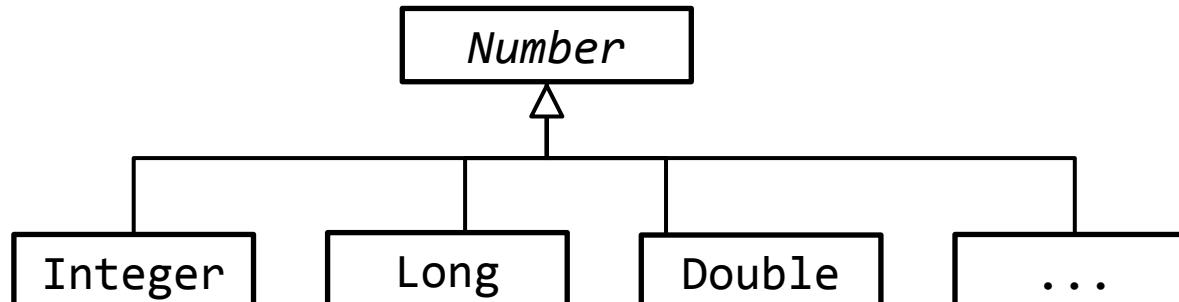
LineChart



- LineChart constructor parameters are the two Axis:
`LineChart(Axis<XType> xAxis, Axis<YType> yAxis)`
- The *abstract* class `Axis` is defined with generics.
Concrete subclasses: CategoryAxis, NumberAxis



- `java.lang.Number` is an abstract superclass for numeric types:



LineChart example 1/2

```
public class LineChartView extends BorderPane {  
    private LineChart<String, Number> lineChart;  
    private static final Double[] MINIMA = {0.7, 0.7, 3.1, 5.3,  
        9.2, 11.9, 14.0, 13.6, 10.9, 7.8, 4.1, 1.6};  
    private static final Double[] MAXIMA = {5.7, 6.6, 10.4, 14.2,  
        18.1, 20.6, 23.0, 22.6, 19.0, 14.7, 9.5, 6.1};
```

```
public LineChartView() {  
    initialiseNodes();  
    layoutNodes();  
}
```

```
private void layoutNodes() {  
    setCenter(this.lineChart);  
}
```

```
private void initialiseNodes() {  
    lineChart = new LineChart<>(new CategoryAxis(),  
        new NumberAxis());
```

```
XYChart.Series<String, Number> series1 =  
    new XYChart.Series<>();
```

```
series1.setName("Average minima");
```

Putting Strings (X)
horizontally and
numbers (Y) vertically

We will use these numbers
later to make Series of Data
pairs

CategoryAxis is
an axis of Strings.

// Continued on next slide...

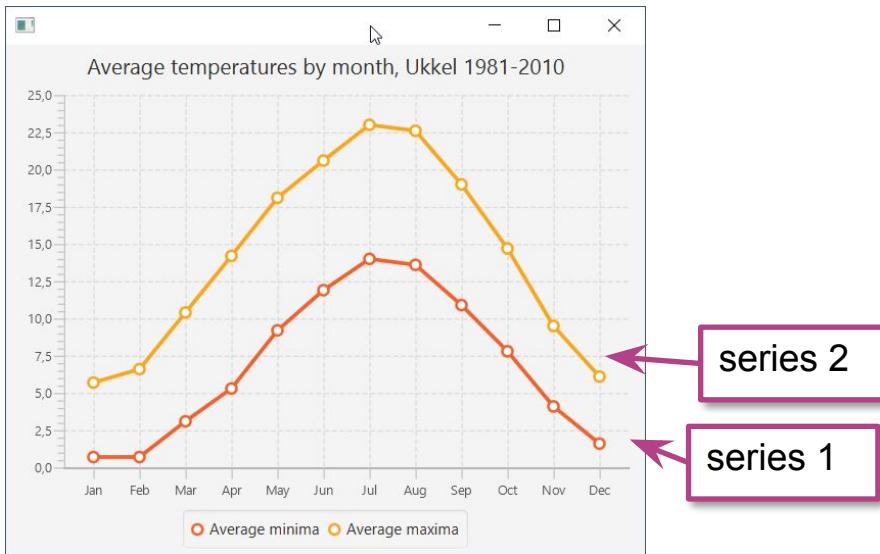
LineChart example 2/2

```
// Continued from previous slide...
```

```
XYChart.Series<String, Number> series2 =  
    new XYChart.Series<>();  
series2.setName("Average maxima");  
for (int i = 0; i < MINIMA.length; i++) {  
    String month = Month.of(i + 1)  
        .getDisplayName(TextStyle.SHORT, Locale.ENGLISH);  
    series1.getData().add(new XYChart.Data<>(month,  
        MINIMA[i]));  
    series2.getData().add(new XYChart.Data<>(month,  
        MAXIMA[i]));  
}  
lineChart.getData().addAll(series1, series2);  
}
```

Every line is a **series**. Types match
LineChart<String, Number>

Make a month/temperature
(String/Number) data pair
and add it to a Series.



series 2

series 1

Charts and MVP

- Chart examples are kept simple, they are not MVP
- For a good MVP design
 - data should come from the model!
 - package-private getter for Chart in View
 - Presenter gets data from model and shows in View:

Presenter.java

```
//...
private void updateView() {
    Map<String, Double> data = this.model.getData();
    ObservableList<PieChart.Data> pieChartData =
        FXCollections.observableArrayList();
    for (String key : data.keySet()) {
        pieChartData.add(new PieChart.Data(key,
            data.get(key)));
    }
    this.view.getPieChart().setData(pieChartData);
}
//...
```

In `updateView` the presenter refreshes with current model data.

Model provides data (e.g. as a `Map`)

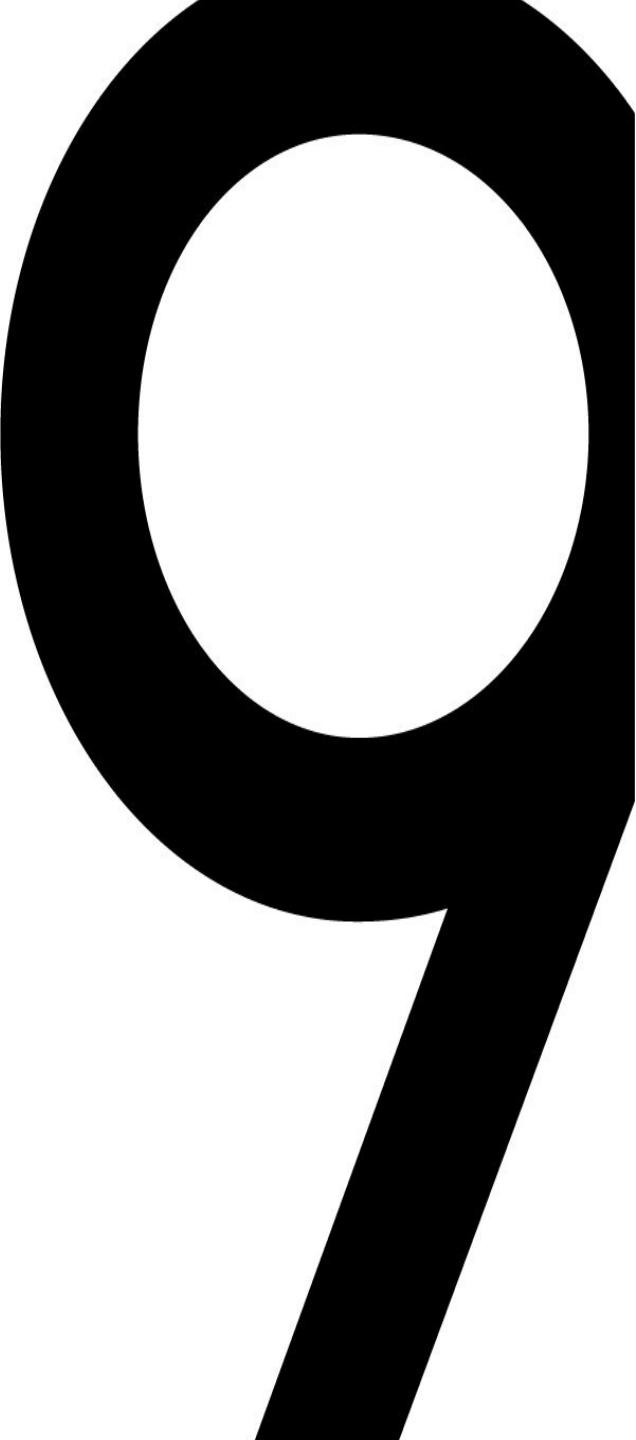
Set View Data

Exercises



- Complete exercises Java 5.05-5.06

Exceptions and MVP



Exceptions and MVP



Model



Presenter

Exceptions in Model



- Make your model “bullet-proof”. Implement the necessary checks. It shouldn’t just crash...
- Use existing exception classes where possible (i.e., `IllegalArgumentException`)
- You can create custom exceptions for exceptions that are specific to your model. It’s recommended to use unchecked exceptions.
 - Use clear exception message to easily detect the cause
- When calling methods that throw exceptions, you can catch them and re-throw them or wrap them in one of your custom exceptions.

Exceptions in Model



```
public void doGuess(Combination guess) {  
    if (guess == null) {  
        throw new IllegalArgumentException(  
            "guess should not be null");  
    }  
    if (guesses.size() >= maxNumberOfGuesses) {  
        throw new MastermindException(  
            "Maximum number of guesses reached!");  
    }  
    guesses.add(guess);  
    // other code omitted  
}
```

A custom exception, such as this one, can be used for domain-specific exceptions

IllegalArgumentException
can be used when checking method arguments

Exceptions in Model



```
public class MastermindException extends RuntimeException {  
    public MastermindException(String s) {  
        super(s);  
    }  
  
    public MastermindException(Throwable cause) {  
        super(cause);  
    }  
  
    public MastermindException(String s, Throwable cause) {  
        super(s, cause);  
    }  
}
```

This custom exception is unchecked.
The second and third constructor
can be used to wrap other
exceptions.



Exceptions in Model



```
public void saveGame() throws MastermindException {  
    MastermindSaver saver = new MastermindSaver(this);  
    try {  
        saver.save();  
    } catch (IOException e) {  
        System.err(e.getMessage());  
        throw new MastermindException(e);  
    }  
}
```

Saving a game can cause an IOException. We'll catch it and wrap it in our custom MastermindException.

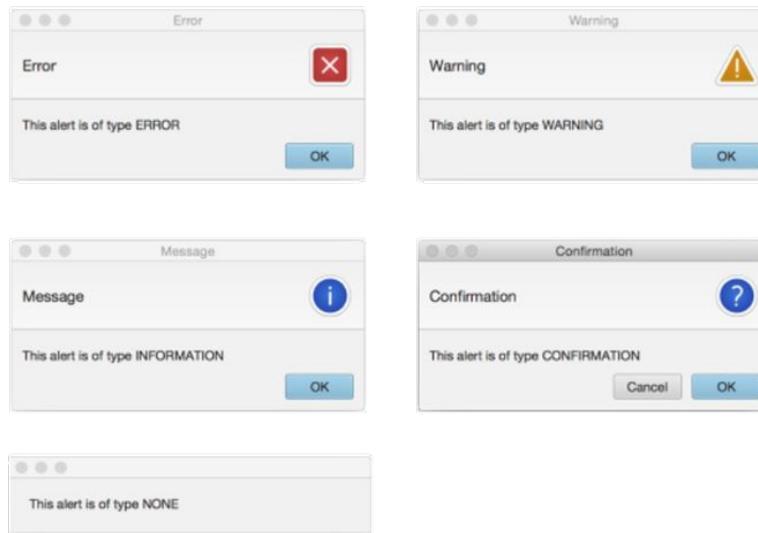
The original error message is written to error output...



Exceptions in Presenter



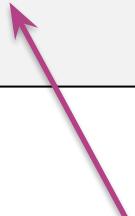
- Try to catch exceptions that are thrown in the model.
- Convert the exception message into a meaningful error message for the user and show it using an Alert.



Exceptions in Presenter



```
view.getMiSave().setOnAction(event -> {
    try {
        model.saveGame();
    } catch (MastermindException e) {
        Alert alert = new Alert(Alert.AlertType.ERROR);
        alert.setTitle("Unable to save:");
        alert.setContentText(e.getMessage());
        alert.showAndWait();
    }
});
```



The Alert class is used to display a meaningful error message ...



1. MVP multi window package structure
2. Start Scene on current Stage
3. Start Scene on new Stage()
4. Alert
5. MVP multi window summary
6. Graphics
7. Background, Node Effects
8. Charts
9. Exceptions and MVP