UNIVERSITY OF WESTMINSTER

INFORMATICS INSTITUTE OF TECHNOLOGY

Software Development II

Lecture 10 – Big O notation and programming methodologies

Content

Big O notation

- Process
- Examples

Software Development Life Cycle

- Requirements gathering
- Design
- Implementation
- Testing
- Documentation
- Maintenance

Programming Methodologies

- The waterfall model
- Agile

More than Java programming

- Version control
- Databases
- Graphical User Interfaces
- Multithreading

Big O notation

Big O notation is a mathematical notation used to analyse the complexity and efficiency of algorithms (e.g., in terms of input size).

Determining the time complexity of a method involves analysing the number of operations performed by the method as a function of the size of its input.

Big O notation

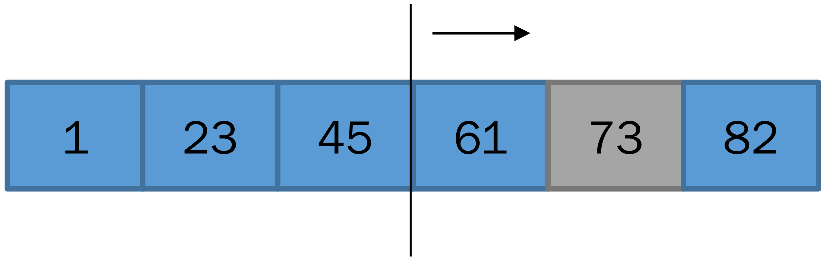
- Big O notation is a system to measure the complexity of an algorithm.
- It analyses the **worst-case**, telling us that the algorithm will always perform equal or better than the worst-case scenario.
- The Big O notation helps us ensure our programs are **efficient**.
- It measures the time (or number of steps) that it takes to complete a problem of size **n**.

Big O notation

- **Constant**:

It always takes the same amount of time (constant), no matter the size of the data: **O(1)**.

Example: `System.out.println(“Hello”);`



- **Logarithmic**:

After each pass, the data/problem is half the size: **O(log N)**.

Example: Binary search (search in a sorted array).

- **Linear**:

If the data size (n) increases by one (n+1), the complexity also increases by one: **O(N).**

Example: Print all elements of an array

Big O notation (cont.)

- **Polynomial**:

If the data size (n) increases by one (n+1), the complexity increases by n: **$O(N^2)$**.

Example: A loop of size n inside another loop of size n (nested loop).

- **Exponential**:

An algorithm that doubles when we increase the data size by one **$O(2^n)$**.

Example: Recursive function with two calls such as Fibonacci with recursion.

Big O notation

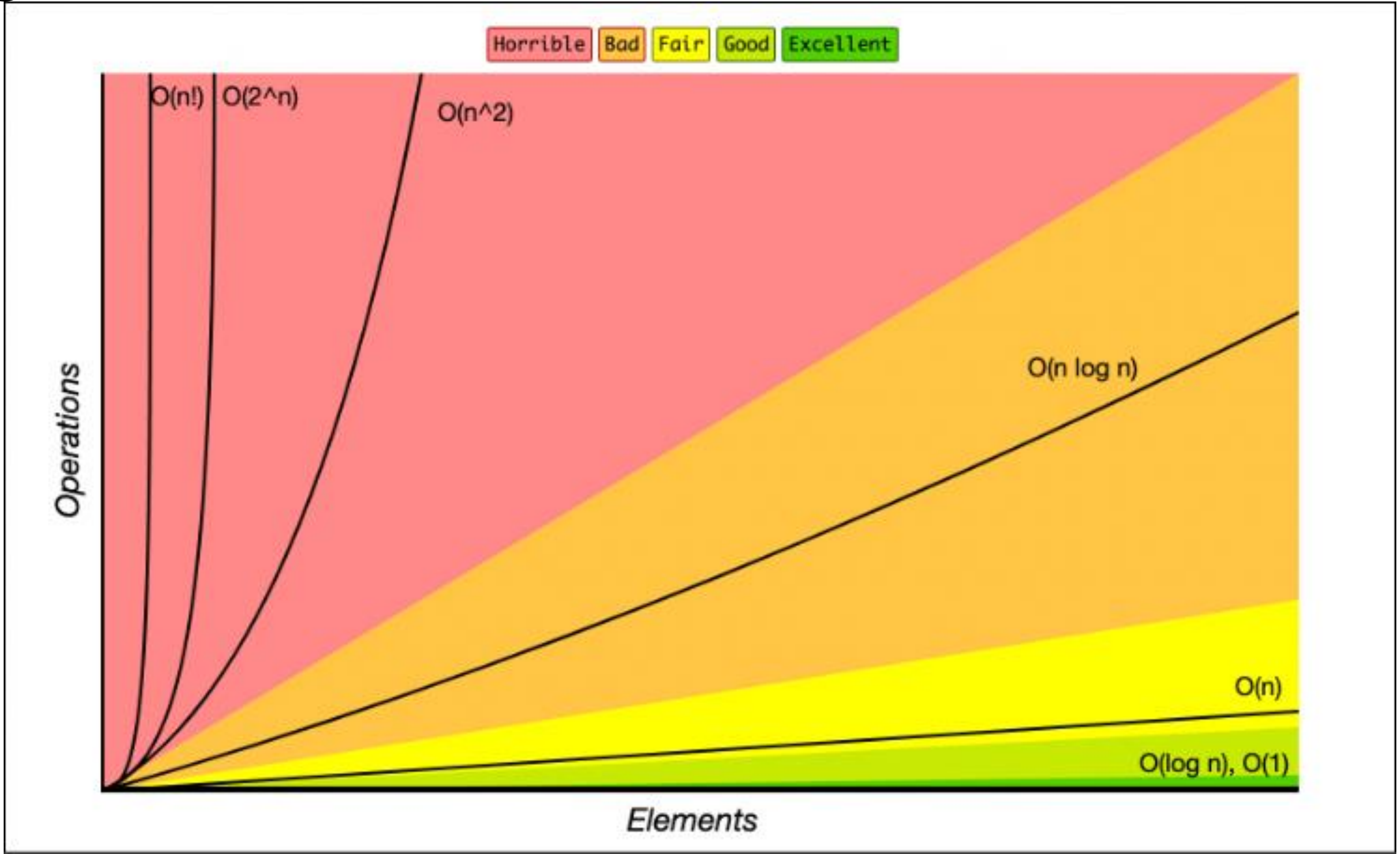


Image source: www.freecodecamp.org

Big O notation: process

1. **Identify** the dominant operation: look for the operation that contributes the most to the overall runtime of the method.
2. **Count** the number of operations: determine how many times the dominant operation is executed in terms of the input size.
3. Express in terms of **input size**: Express the number of operations as a function of the input size.
4. Determine the **Big O notation**: Identify the Big O notation that describes the growth rage of the method as the input size approaches infinity.

Big O notation: Example 1

```
for(int i = 0; i < array.length; i++){
  for(int j = 0; j < array.length; i++){
    if (array[i] < array[j]){
      temp = array[i];
      array[i] = array[j];
      array[j] = array[i];
    }
  }
}
```

O(N)

O(N)

O(1)

O(1)

O(1)

O(1)

O(1)

O(N)

O(N) x O(N)
= O(N^2)

Big O notation: example 2

```java
// Method to find the maximum element in an array
public static int findMax(int[] arr) {
    if (arr == null || arr.length == 0) {
        throw new IllegalArgumentException("Array must not be empty or null");
    }

    int max = arr[0]; // Initialise max to the first element

    // Loop through the array to find the maximum element
    for (int i = 1; i < arr.length; i++) {
        if (arr[i] > max) {
            max = arr[i]; // Update max if current element is greater
        }
    }

    return max;
}
```

1 Dominant operation

2 Number of operations

3 Input size: n

4 O(n)

Which BubbleSort implementation is more efficient?

```
public static void BubbleSort(int[] array) {
    int n = array.length;
    for (int i = 0; i < n - 1; i++) {
        for (int j = 0; j < n - i - 1; j++) {
            if (array[j] > array[j + 1]) {
                // Swap array[j] and array[j+1]
                int temp = array[j];
                array[j] = array[j + 1];
                array[j + 1] = temp;
            }
        }
    }
}
```

Best $O(n^2)$
Average $O(n^2)$
Worst $O(n^2)$

```
private static void BubbleSort(int[] array) {
    int bottom = array.length - 2;
    int temp;
    boolean exchanged = true;

    while (exchanged) {
        exchanged = false;
        for (int i = 0; i <= bottom; i++) {
            if (array[i] > array[i + 1]) {
                temp = array[i];
                array[i] = array[i + 1];
                array[i + 1] = temp;
                exchanged = true;
            }
        }
        bottom--;
    }
}
```

Best $O(n)$
Average $O(n^2)$
Worst $O(n^2)$

SOFTWARE DEVELOPMENT LIFE CYCLE

Software Development Life Cycle (SDLC)

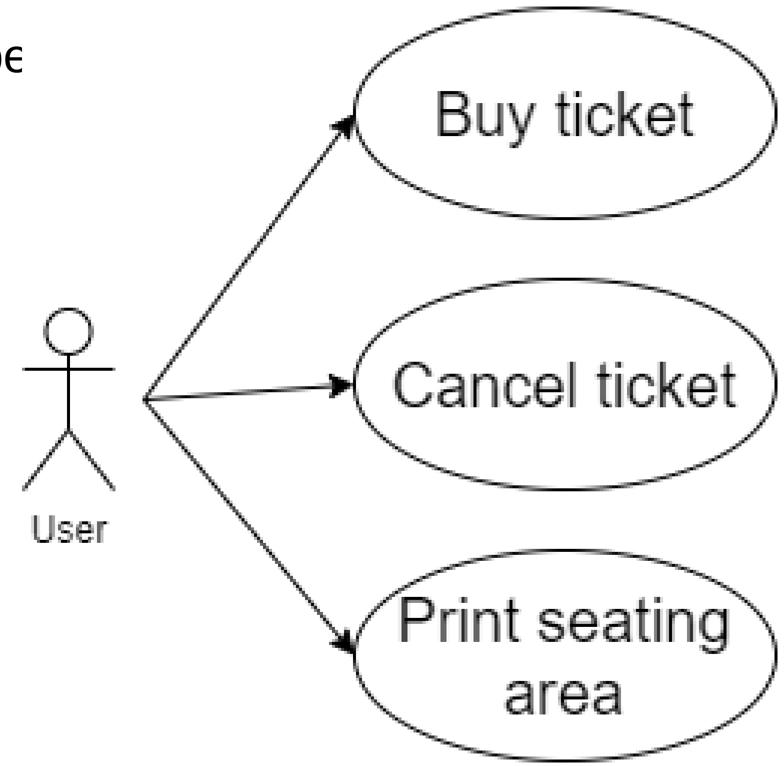
- Programming is not only about writing a Java program.
- Programming is a process:
  - Gather requirements / analysis
  - Design
  - Implementation
  - Testing
  - Deployment
  - Maintenance

The software life cycle is typically iterative, meaning that the stages may be revisited and refined as needed.

In this lecture, we will explore programming methodologies with the task described in the coursework: Theatre.

Requirements gathering

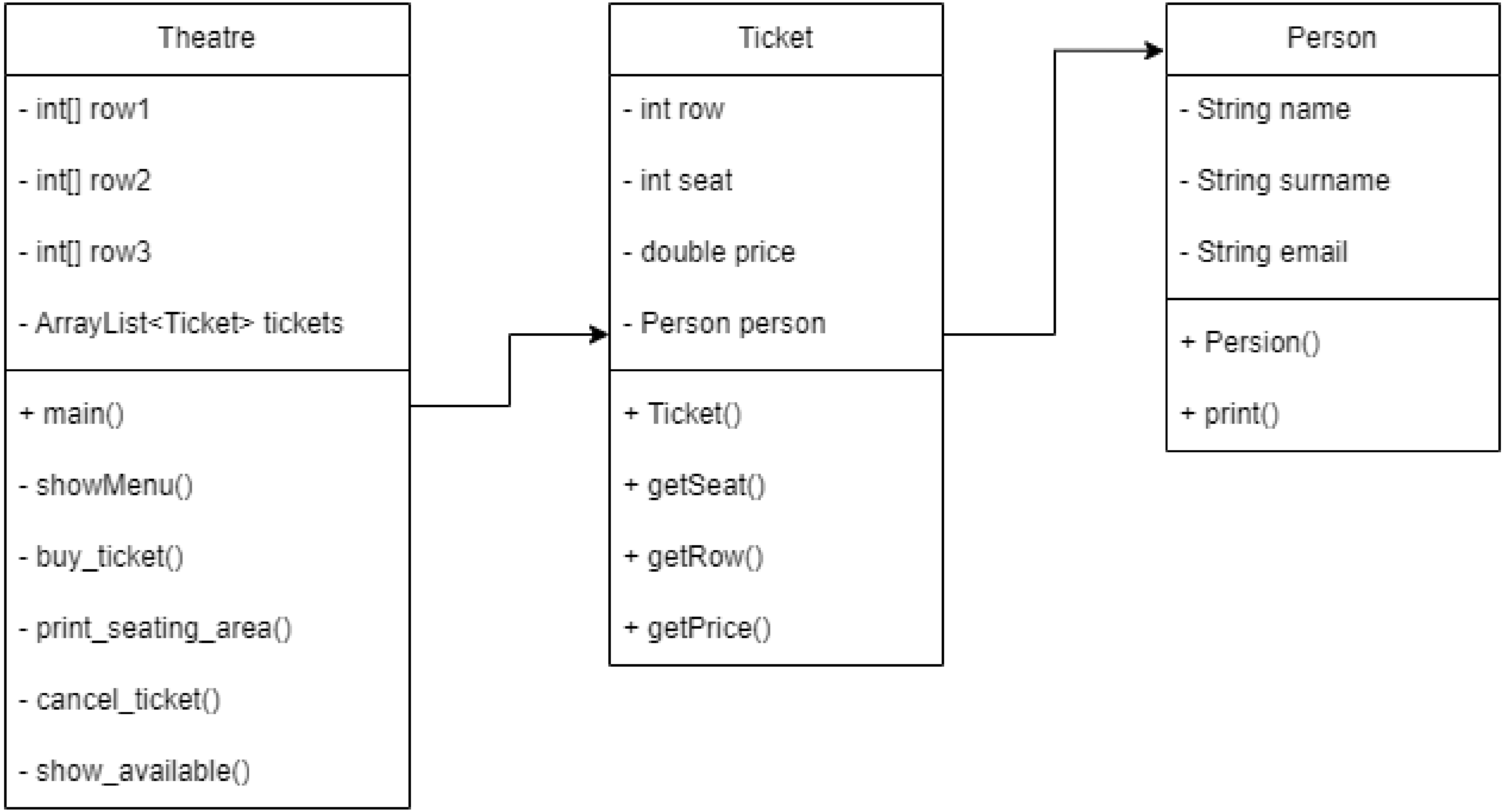
- The requirements for the software are gathered and analysed.
- Determine what the software should do and how it should be



Design

Class diagram:

- Design the architecture:
  - Structure
  - Modules
  - Components
  - Methods
  - Classes
- UML diagrams



Implementation

- Implement the software according to the specifications.
- Programming, testing and debugging

Testing

- Software is tested to:

1. Ensure that the requirements are satisfied.
2. There are not errors in the program.

- Covered in lecture week 5

White box testing, black box testing, integration testing…

Documentation

Software documentation is a set of documents that describe the software design and functionalities such as:

- Requirement documents
- Design documents
- Technical documentation
- User manuals
- Test plans

- Documentation is important because:
  - It provides a shared understanding of the software design, implementation and functionality.
  - Helps to maintain the software
  - Provides end-users information useful to understand how to use the software.

Javadoc

```java
/**
* Method that prints person's surname, name and email.
*/
public void print() {
System.out.println("Surname, name (email): " + surname + ", " + name + " (" + email + ")");
}
```

Function documentation

All Methods	Instance Methods	Concrete Methods

Modifier and Type	Method	Description
void	print()	Method that prints person's surname, name and email.

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

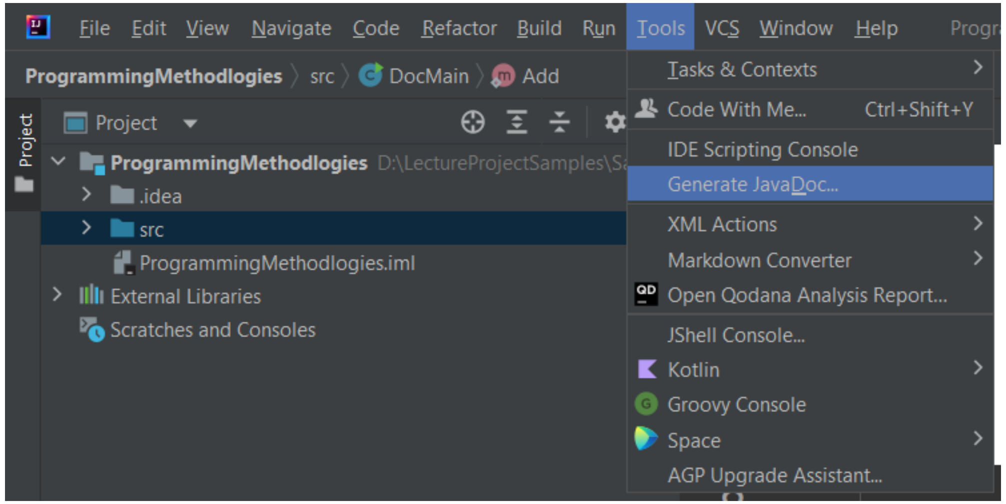
Method Details

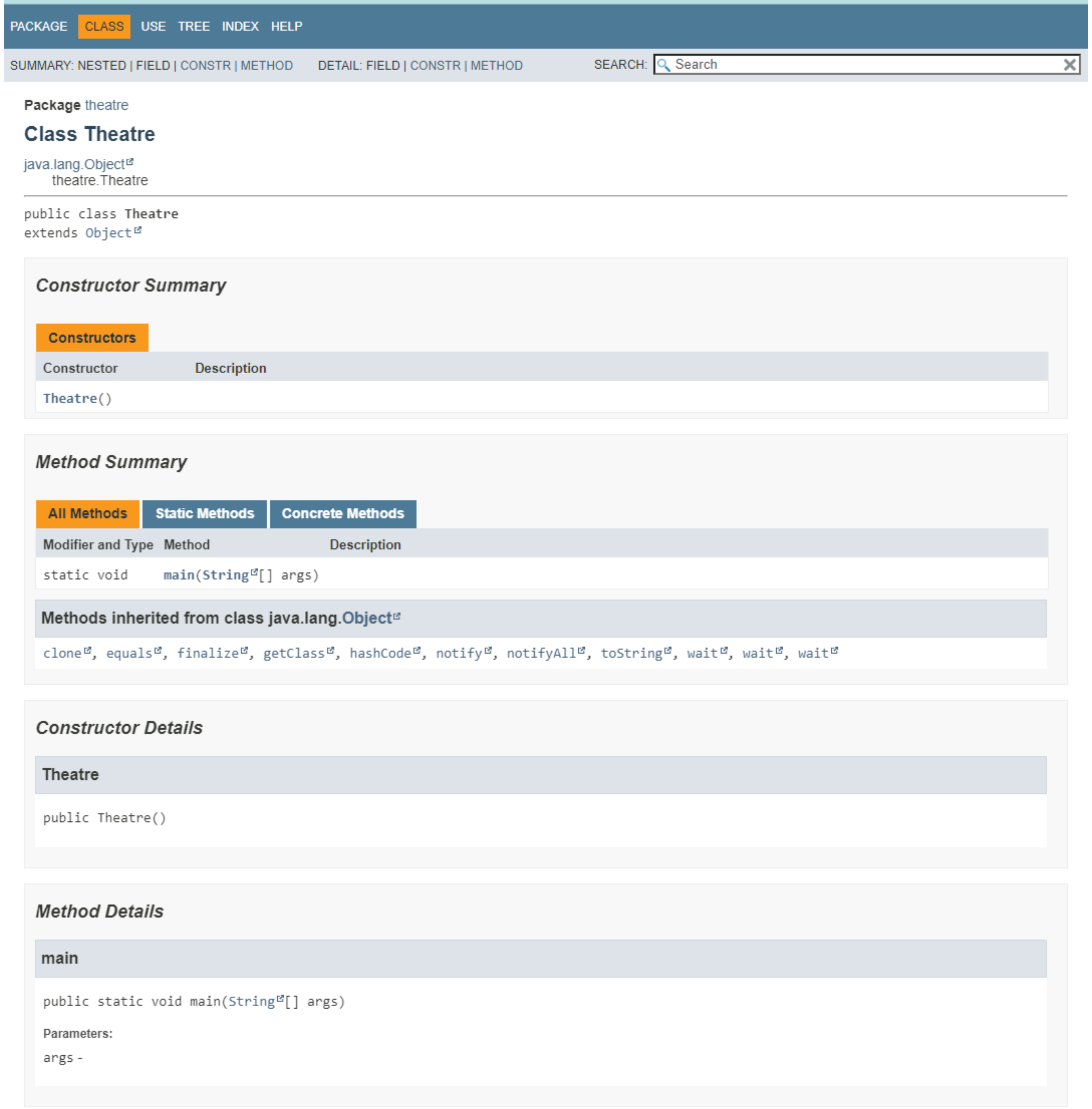
print

`public void print()`

Method that prints person's surname, name and email.

Javadoc with IntelliJ





Javadoc with IntelliJ

Package theatre

Class Ticket

java.lang.Object
theatre.Ticket

```
public class Ticket
extends Object
```

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
double	get_price()	
int	get_row()	
int	get_seat()	
void	print()	

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Details

print

`public void print()`

get_row

`public int get_row()`

get_seat

`public int get_seat()`

get_price

`public double get_price()`

PACKAGE CLASS USE TREE INDEX HELP

SUMMARY: NESTED | FIELD | CONSTR | METHOD DETAIL: FIELD | CONSTR | METHOD SEARCH: Search

Package theatre

Class Person

java.lang.Object
theatre.Person

```
public class Person
extends Object
```

Method Summary

All Methods | Instance Methods | Concrete Methods

Modifier and Type	Method	Description
void	print()	

Methods inherited from class java.lang.Object

clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait

Method Details

print

`public void print()`

Maintenance

- Maintenance can include several tasks:
  - Add new features
  - Correct errors and bugs
  - Improve performance

- Debugging is a key strategy to find and fix bugs.

PROGRAMMING METHODOLOGIES

Programming methodologies

- Methodology to develop software.
- There are several programming methodologies that are commonly used in software development (programming):
  - The waterfall model
  - Agile

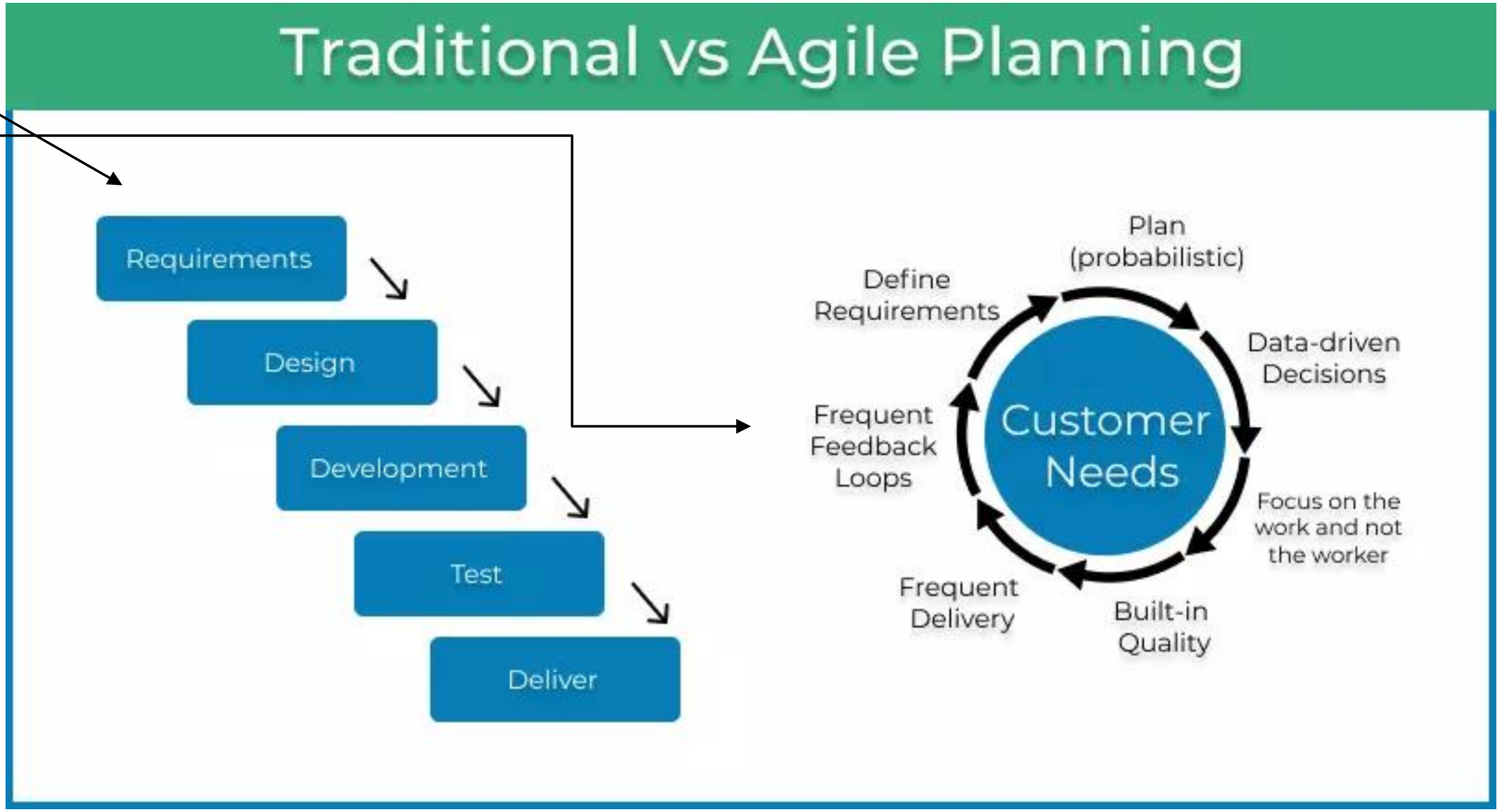


Image source: https://kanbanize.com/agile/project-management/planning

Programming methodologies: The waterfall model

- Sequential approach that involves the following phases:

Programming methodologies: Agile

- Iterative process with shorter development cycles, frequent testing and continuous feedback from the user.

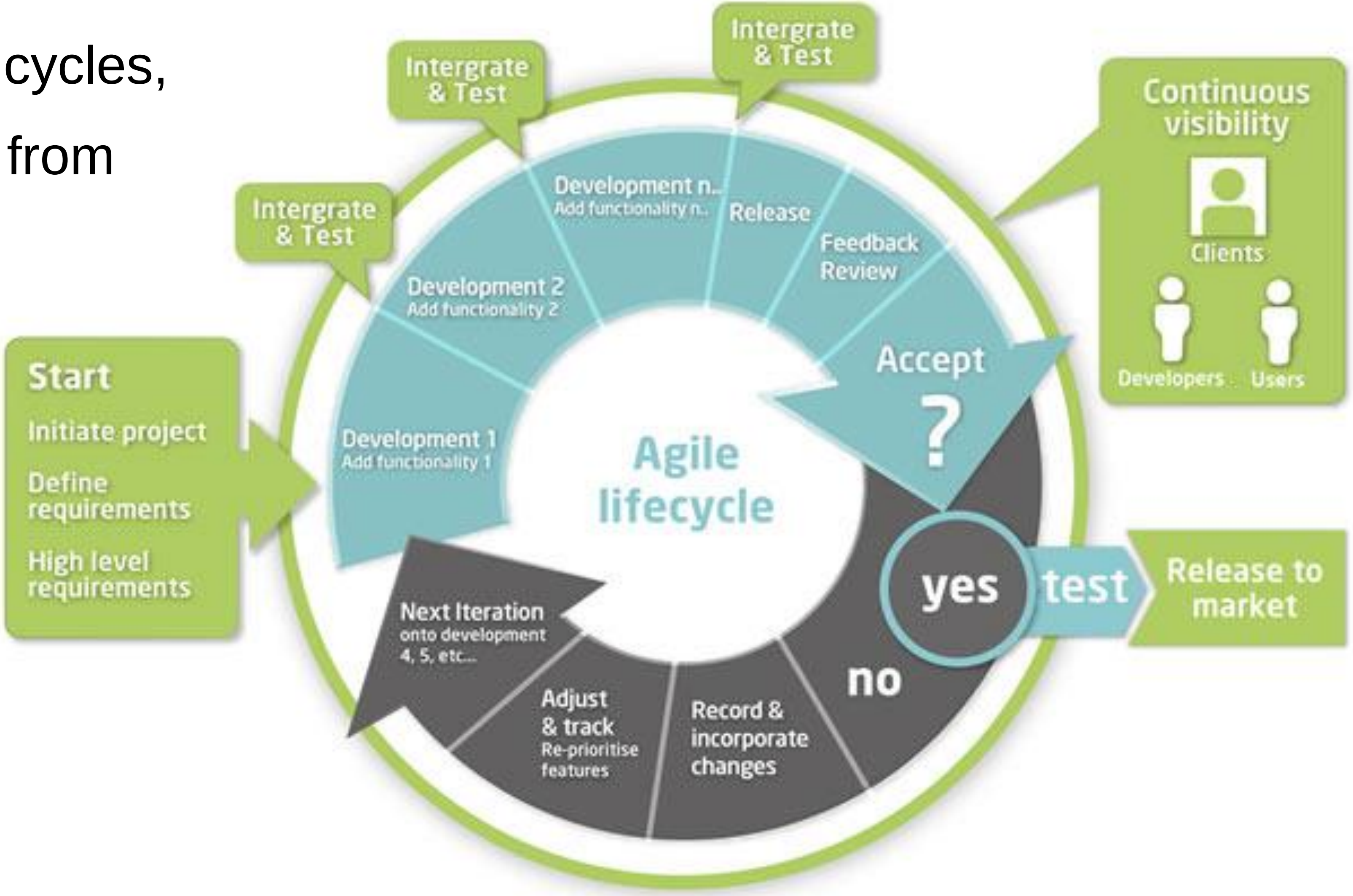


Image source: https://artdriver.com/blog/what-is-agile-project-management

COMMUNICATION: THE BIGGEST CHALLENGE

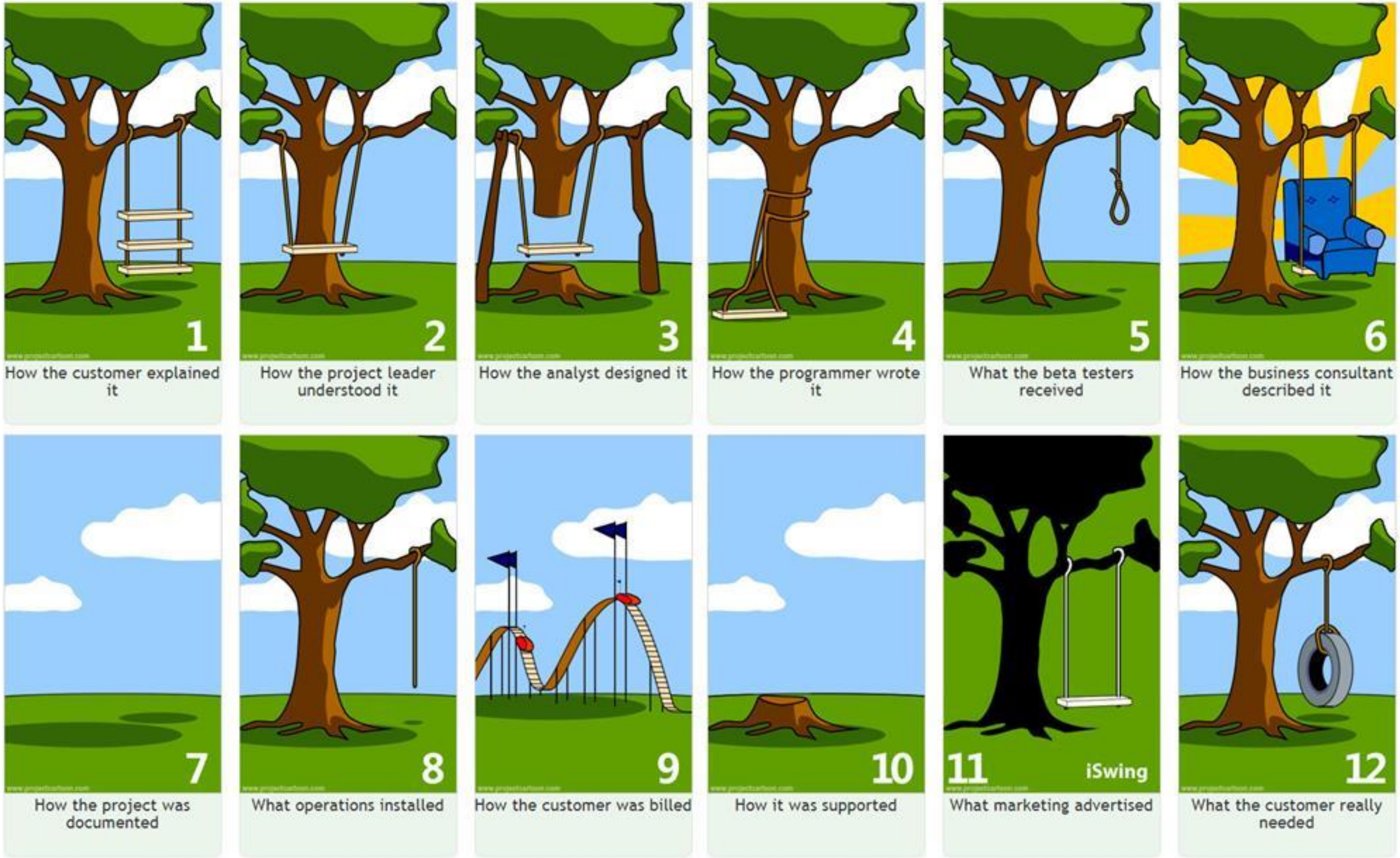


Image source: https://www.zentao.pm/blog/tree-swing-project-management-tire-analogy-426.html

Thank You !!