Lesson 8 – Optional, DateTime API, Gradle

Lesson Goals

- DateTime API
- Optional class
- Gradle

Date Time API

Java Date Time API

- Since Java 8
- Easy for understanding
- Flexible API
- Immutable (i.e. thread safe)
- Can work with Zoned time

Most used classes in “java.time.*” package

- LocalDate – represents **a date in ISO format (yyyy-MM-dd) without time.**
- LocalTime - represents **time without a date.**
- LocalDateTime is used to represent **a combination of date and time.**

Working with LocalDate

```
LocalDate localDate = LocalDate.now();
LocalDate.of(2015, Month.FEBRUARY, 20);
LocalDate.parse("2015-02-20");
LocalDate tomorrow = LocalDate.now().plusDays(1);
LocalDate previousMonthSameDay = LocalDate.now().minus(1, ChronoUnit.MONTHS);
DayOfWeek sunday = LocalDate.parse("2016-06-12").getDayOfWeek();
int twelve = LocalDate.parse("2016-06-12").getDayOfMonth();
boolean leapYear = LocalDate.now().isLeapYear();
```

Working with LocalTime

```
LocalTime now = LocalTime.now();
LocalTime sixThirty = LocalTime.parse("06:30");
LocalTime sixThirtyParsed = LocalTime.of(6, 30);
LocalTime sevenThirty = LocalTime.parse("06:30").plus(1, ChronoUnit.HOURS);
int six = LocalTime.parse("06:30").getHour();
boolean isBefore = LocalTime.parse("06:30").isBefore(LocalTime.parse("07:30"));
```

Working with LocalDateTime

```
LocalDateTime.now();
LocalDateTime.of(2015, Month.FEBRUARY, 20, 06, 30);
LocalDateTime.parse("2015-02-20T06:30:00");
LocalDateTime.of(LocalDate.now(), LocalTime.now());
```

Date and Time Formatting

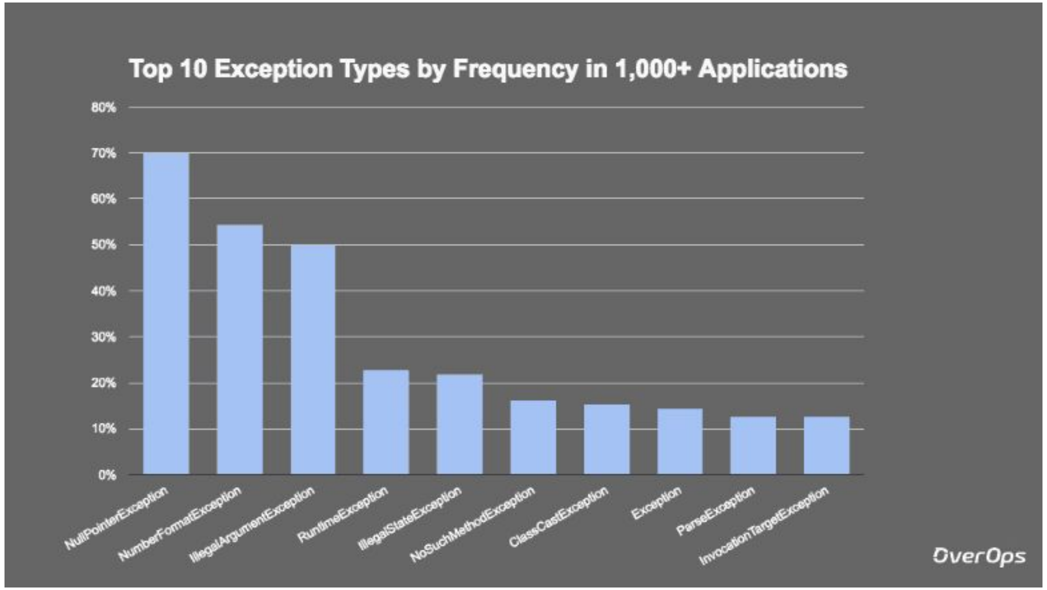
- Common format is ISO-8601 – ‘2011-12-03’ or ‘2011-12-03+01:00’.
- LocalDateTime localDateTime = LocalDateTime.*of*(2015, Month.***JANUARY***, 25, 6, 30);
- String localDateString = localDateTime.format(DateTimeFormatter.***ISO_DATE***);
- localDateTime.format(DateTimeFormatter.*ofPattern*(**"yyyy/MM/dd"**));

Billion-Dollar mistake

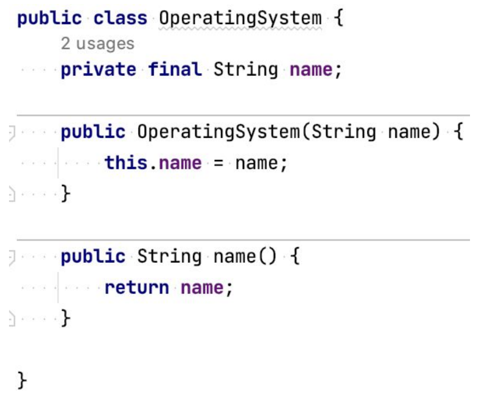
NPE aka “Billion-Dollar mistake”

- Tony Hoare the creator of the null referen
- Created while designing ALGOL W
- Meant to model the absence of a value
- “... simply because it was so easy to implement”
- The cause of the Null Pointer Exceptions

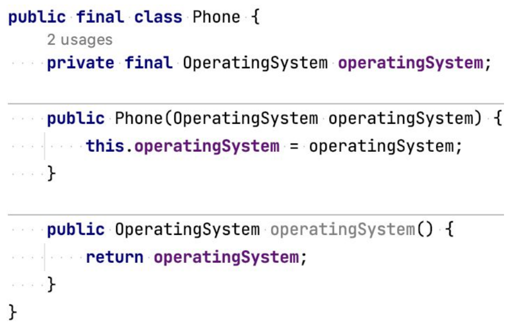
Most frequent exception



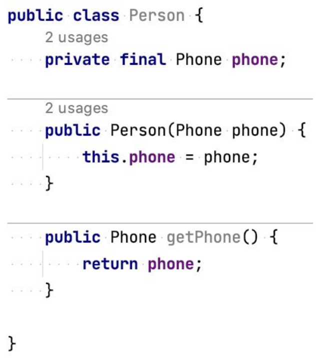
NPE example



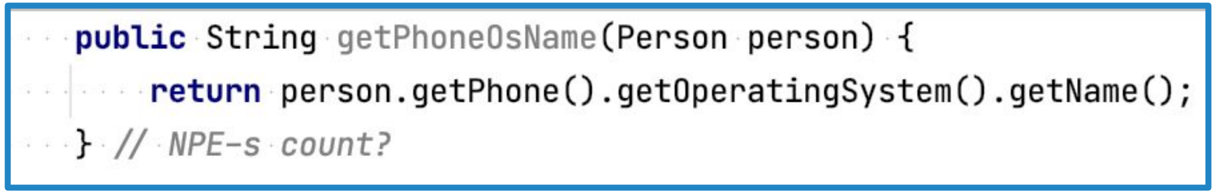
```java
public class OperatingSystem {
    2 usages
    private final String name;

    public OperatingSystem(String name) {
        this.name = name;
    }

    public String name() {
        return name;
    }

}
```



```java
public final class Phone {
    2 usages
    private final OperatingSystem operatingSystem;

    public Phone(OperatingSystem operatingSystem) {
        this.operatingSystem = operatingSystem;
    }

    public OperatingSystem operatingSystem() {
        return operatingSystem;
    }
}
```

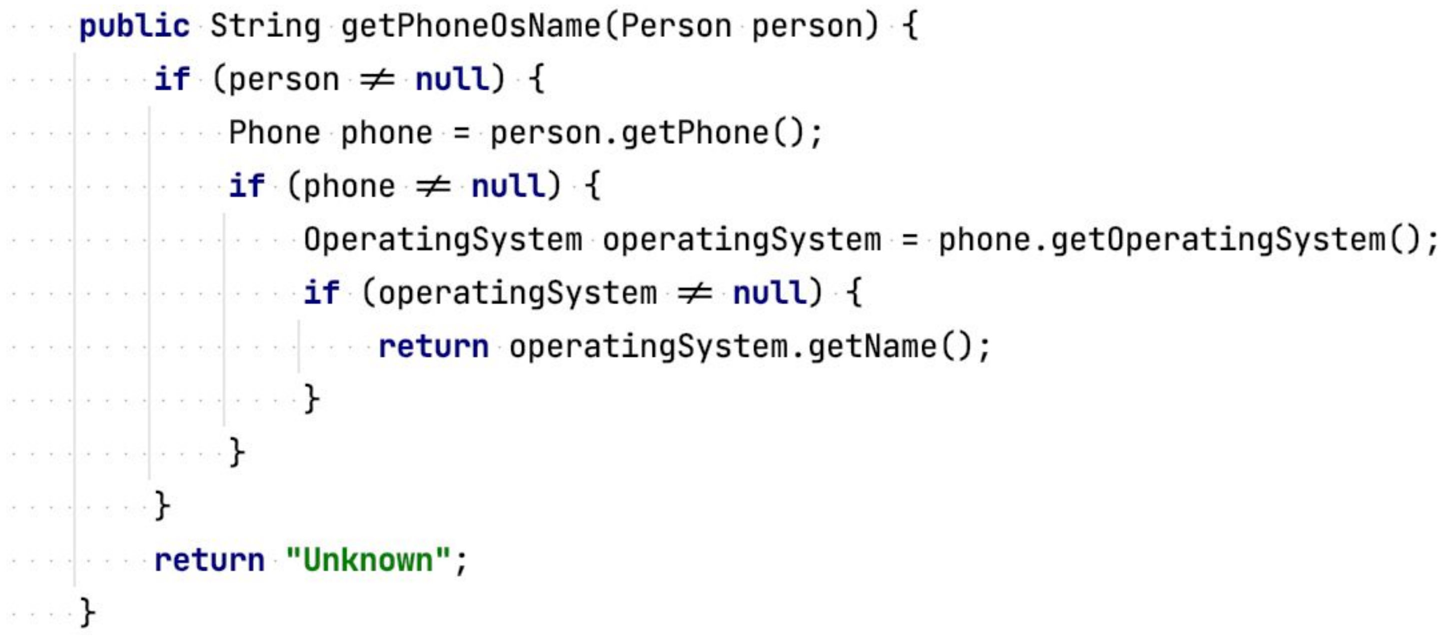


```java
public class Person {
    2 usages
    private final Phone phone;

    2 usages
    public Person(Phone phone) {
        this.phone = phone;
    }

    public Phone getPhone() {
        return phone;
    }

}
```



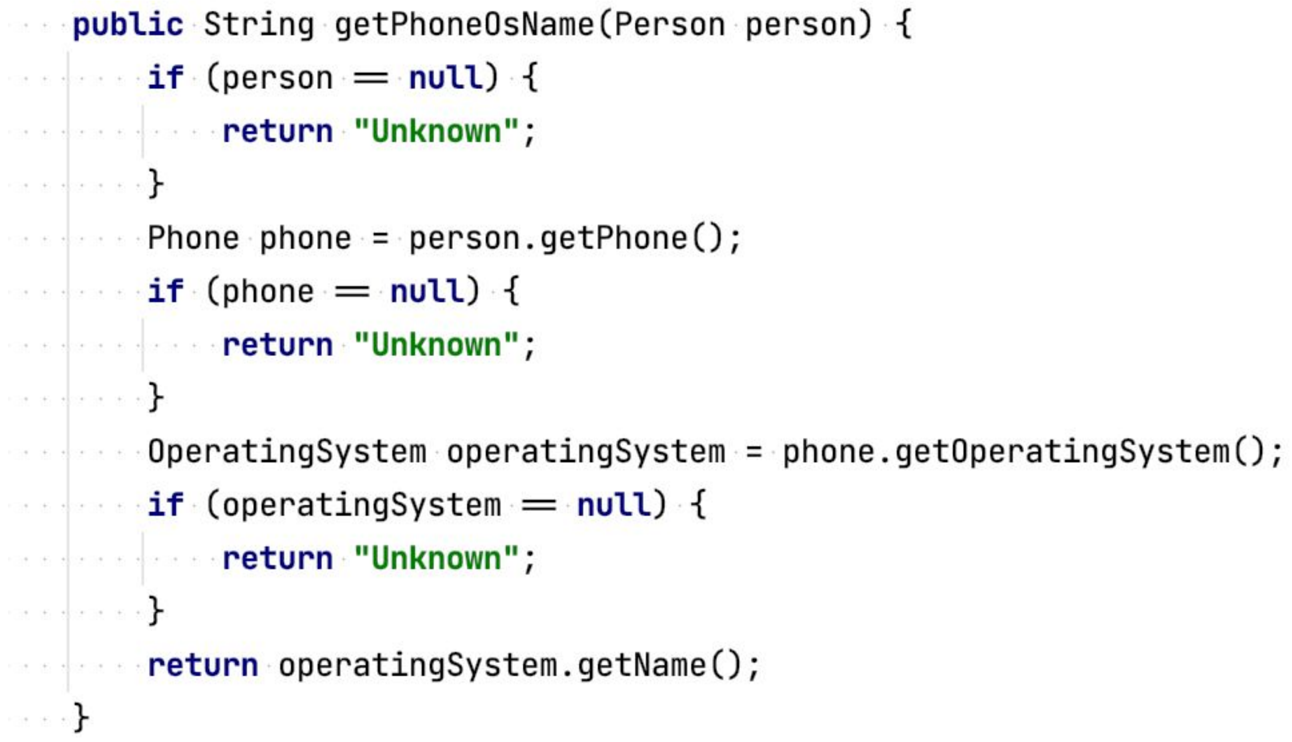
```java
    public String getPhoneOsName(Person person) {
        return person.getPhone().getOperatingSystem().getName();
    } // NPE-s count?
```

Attempt 1: Null checks



```java
public String getPhoneOsName(Person person) {
    if (person ≠ null) {
        Phone phone = person.getPhone();
        if (phone ≠ null) {
            OperatingSystem operatingSystem = phone.getOperatingSystem();
            if (operatingSystem ≠ null) {
                return operatingSystem.getName();
            }
        }
    }
    return "Unknown";
}
```

Attempt 2: Multiple Exits (Fail/Return First pattern)



```java
public String getPhoneOsName(Person person) {
    if (person == null) {
        return "Unknown";
    }
    Phone phone = person.getPhone();
    if (phone == null) {
        return "Unknown";
    }
    OperatingSystem operatingSystem = phone.getOperatingSystem();
    if (operatingSystem == null) {
        return "Unknown";
    }
    return operatingSystem.getName();
}
```

The problem – NullPointerException

-
  ```
  var operatingSystem = new OperatingSystem("Symbian");
  ```

-
  ```
  var operatingSystem = new OperatingSystem(null);
  ```

-
  ```
  var phone = new Phone(operatingSystem);
  var person = new Person(phone);

  String phoneOsName = getPhoneOsName(person);
  println(phoneOsName.length());
  ```

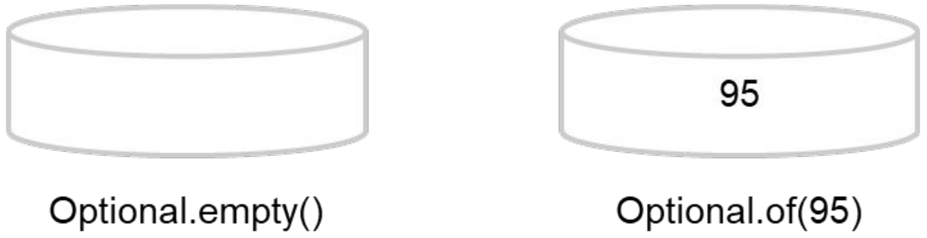
-
  ```
  Exception in thread "main" java.lang.NullPointerException
  ```

ARE YOU REALLY SURE THAT THIS VARIABLE CAN NEVER EVER BE NULL?
OF COURSE!!!

Optional – for the rescue

Optional – for the rescue

- New class located at java.util.Optional<T>
- Encapsulates optional values
- It wraps the value if it's there, and is empty if it isn't
  - Optional.empty()
- Signals that a missing value is acceptable
- We can't easy operate with null, but we can operate with Optional.empty()



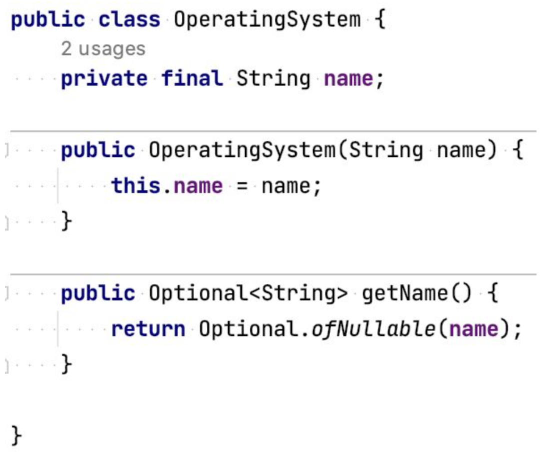
Warning before we progress

- Optionals are not intended to replace all null references
- Design more comprehensible APIs
- Forces caller to deal with the absence of a value

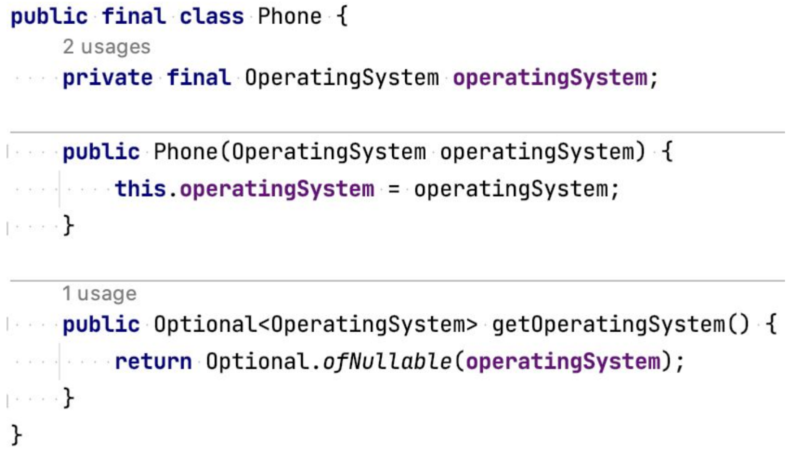
How to create Optional objects

- Empty Optional
  - `Optional<T> opt = Optional.empty();`
  - `Optional<Phone> phoneOpt = Optional.empty();`
- Optional from a non-null value
  - `Optional<T> opt = Optional.of(T value);`
  - `Optional<Phone> phoneOpt = Optional.of(phone);`
- Optional from a null value
  - `Optional<T> opt = Optional.ofNullable(T value);`

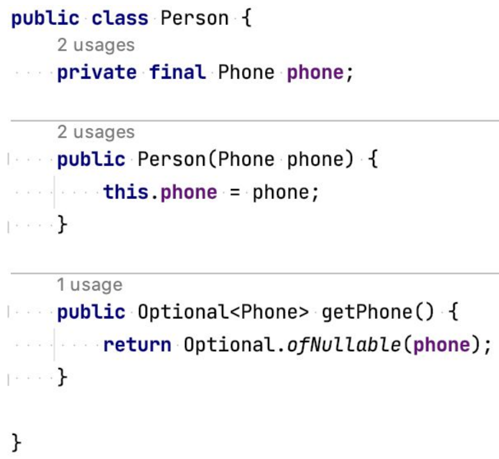
Attempt 3: Refactoring with Optional



```java
public class OperatingSystem {
    2 usages
    private final String name;

    public OperatingSystem(String name) {
        this.name = name;
    }

    public Optional<String> getName() {
        return Optional.ofNullable(name);
    }

}
```

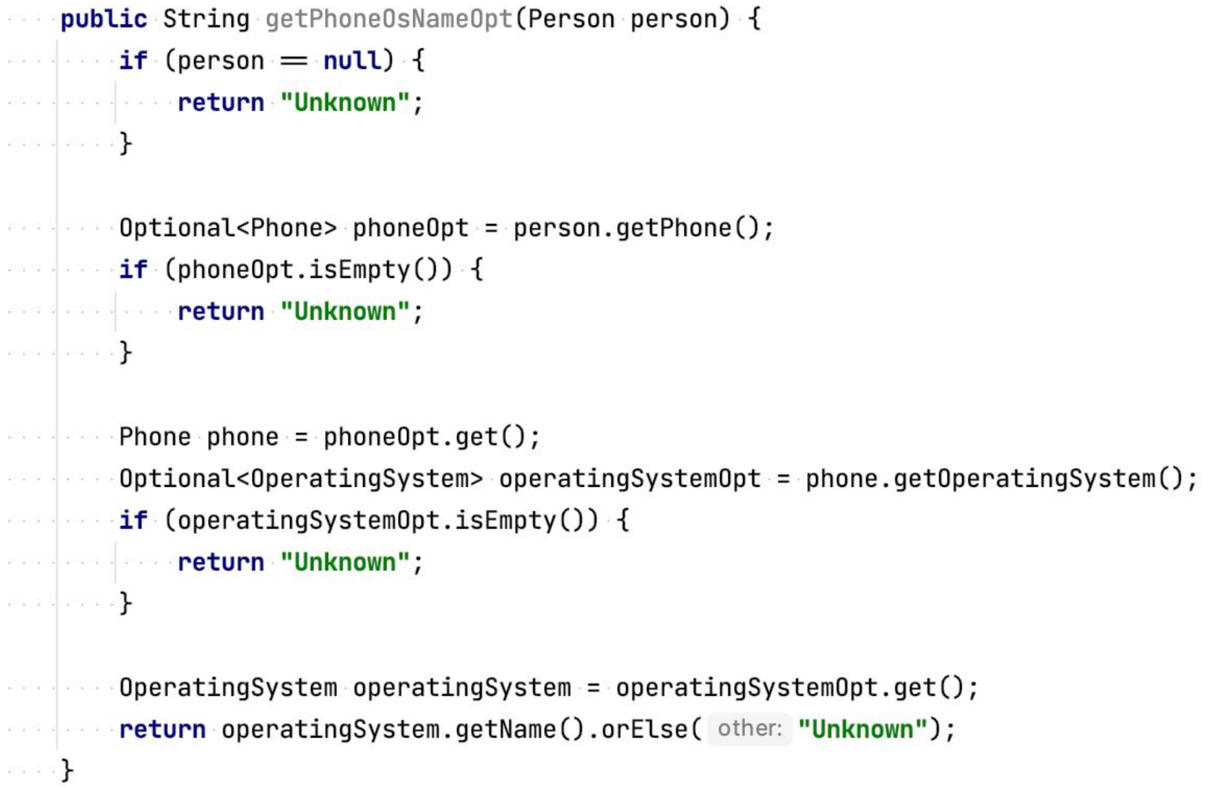


```java
public final class Phone {
    2 usages
    private final OperatingSystem operatingSystem;

    public Phone(OperatingSystem operatingSystem) {
        this.operatingSystem = operatingSystem;
    }

    1 usage
    public Optional<OperatingSystem> getOperatingSystem() {
        return Optional.ofNullable(operatingSystem);
    }
}
```



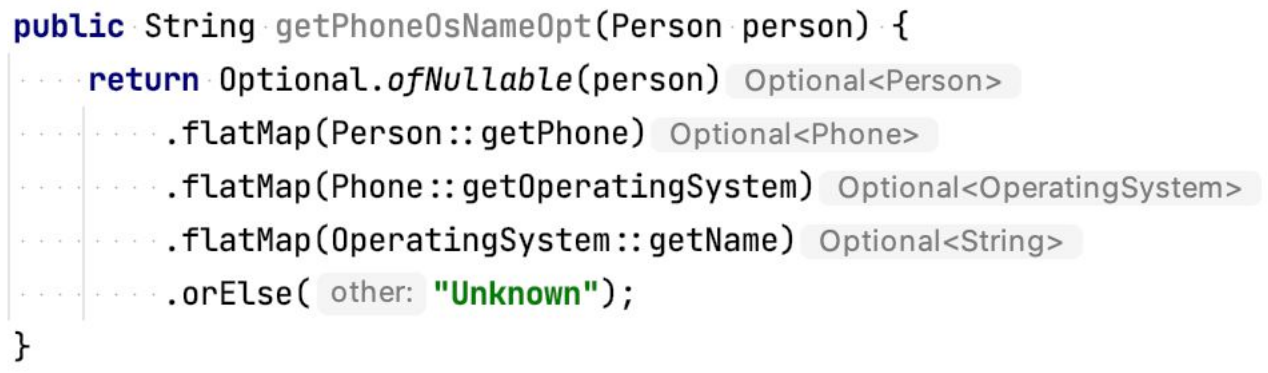
```java
public class Person {
    2 usages
    private final Phone phone;

    2 usages
    public Person(Phone phone) {
        this.phone = phone;
    }

    1 usage
    public Optional<Phone> getPhone() {
        return Optional.ofNullable(phone);
    }

}
```

Attempt 3: Optional is easy 😈



```java
public String getPhoneOsNameOpt(Person person) {
    if (person == null) {
        return "Unknown";
    }

    Optional<Phone> phoneOpt = person.getPhone();
    if (phoneOpt.isEmpty()) {
        return "Unknown";
    }

    Phone phone = phoneOpt.get();
    Optional<OperatingSystem> operatingSystemOpt = phone.getOperatingSystem();
    if (operatingSystemOpt.isEmpty()) {
        return "Unknown";
    }

    OperatingSystem operatingSystem = operatingSystemOpt.get();
    return operatingSystem.getName().orElse( other: "Unknown");
}
```

Attempt 4: Optional is easy 😇



```
public String getPhoneOsNameOpt(Person person) {
    return Optional.ofNullable(person) Optional<Person>
            .flatMap(Person::getPhone) Optional<Phone>
            .flatMap(Phone::getOperatingSystem) Optional<OperatingSystem>
            .flatMap(OperatingSystem::getName) Optional<String>
            .orElse( other: "Unknown");
}
```

Optional Methods

```java
T get();
boolean isPresent();
boolean isEmpty();
void ifPresent(Consumer<? super T> action);
void ifPresentOrElse(Consumer<? super T> action, Runnable emptyAction);
Optional<T> filter(Predicate<? super T> predicate);
<U> Optional<U> map(Function<? super T, ? extends U> mapper);
<U> Optional<U> flatMap(Function<? super T, ? extends Optional<? extends U>> mapper);
Optional<T> or(Supplier<? extends Optional<? extends T>> supplier);
Stream<T> stream();
T orElse(T other);
T orElseGet(Supplier<? extends T> supplier);
T orElseThrow();
<X extends Throwable> T orElseThrow(Supplier<? extends X> exceptionSupplier) throws X;
```

Optional Summary

- Null references have been historically introduced to signal absences
- Optional has been introduced to model presence or absence
- Can create Optional objects using factory methods:
  - `Optional.empty`
  - `Optional.of`
  - `Optional.ofNullable`
- Optional supports methods suited for functional programming
- Using Optional forces callers to actively deal with potential absences
- Optional can help you design better APIs

Gradle

What is

Software build process

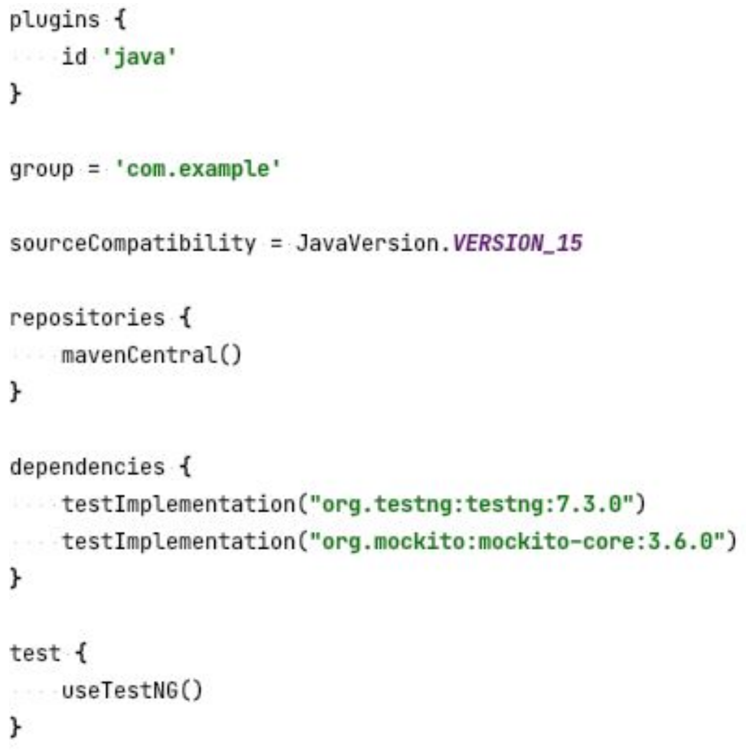
- Building software is hard and require big amount of time!

Software build process

- Building software is hard and require big amount of time!

- Software build process:
  - Develop
  - Test
  - Assemble
  - Deploy
  - Integrate
- Repeat
  - Again and again and again… 🤪

Gradle Project Config Example



```
plugins {
    id 'java'
}

group = 'com.example'

sourceCompatibility = JavaVersion.VERSION_15

repositories {
    mavenCentral()
}

dependencies {
    testImplementation("org.testng:testng:7.3.0")
    testImplementation("org.mockito:mockito-core:3.6.0")
}

test {
    useTestNG()
}
```

Gradle main points

- Can use Wrapper instead separate instalation in system 😍
- Build Scripts are code in Groovy/Kotlin DSL 😎
- Pluggable to work not only with Java 😉
- Flexible build configurations(declarative style) 🙃

- Some times called “StackOverflow driven build system” 🤕

Gradle core concepts

- **Build script:** a build.gradle configuration script supporting one or more projects.
- **Project:** a component that needs to be build. It is made up of one or more tasks.
- **Task:** distinct step required to perform the build.

Gradle core concepts

- **Publication:** the artifact produced by the build process.
- **Dependencies:** tasks and projects depending on each other (internal) or on third-party artifacts (external).
- **Transitive dependencies:** the dependencies of a project may themselves have dependencies.

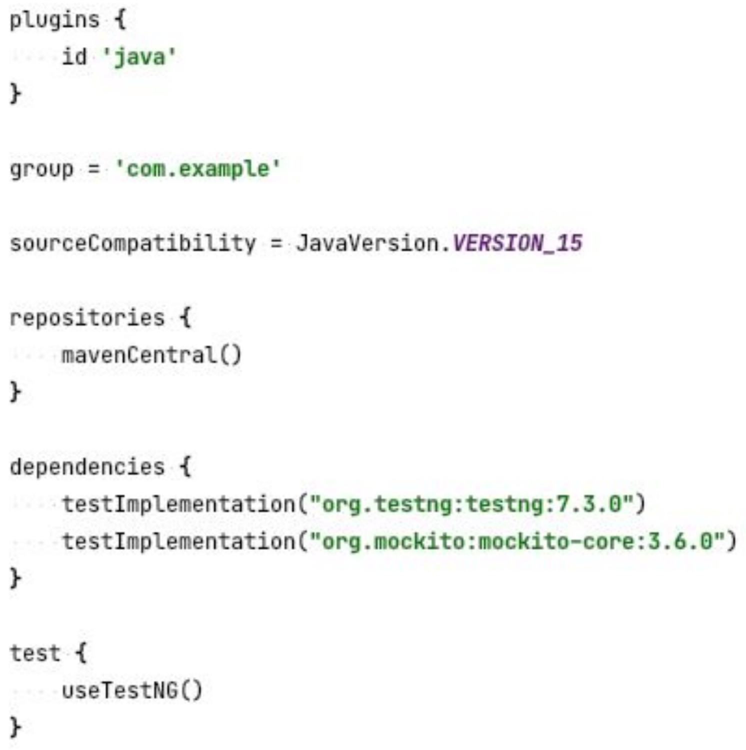
Gradle core concepts

- Repositories: the “places” that hold external dependencies.
- Dependency Graph: shows what depends on what.
- Plugins: external tools that can customize build process

Gradle Plugin power

- Add a task to a project
- Pre-configure these tasks with reasonable defaults
- Add dependency configurations
- Add new properties and methods to existing objects

Gradle Project Config Example



```
plugins {
    id 'java'
}

group = 'com.example'

sourceCompatibility = JavaVersion.VERSION_15

repositories {
    mavenCentral()
}

dependencies {
    testImplementation("org.testng:testng:7.3.0")
    testImplementation("org.mockito:mockito-core:3.6.0")
}

test {
    useTestNG()
}
```

The Build Lifecycle

- Initialization: initialization of the project
- Configuration: configuration of the project (computes the dependencies graph)
- Execution: executes the sequence of build tasks

Multi Module Projects

- Each module/project with its own “build.gradle”
- Each module described in “settings.gradle” file in the root of project folder
- Root project “build.gradle” should contain only configs related to all or subprojects

Gradle Example

1. Show Gradle Tool in IDEA
2. Show Dependency Graph
3. Execution of tasks from command line
4. Multi module project structure
5. Multiple projects configuration (subprojects)
6. JAR building

Homework Task 1. Optional

- Use an Optional as a return type for methods that may return 'null' or throw exception (example input reader methods may return Optional)

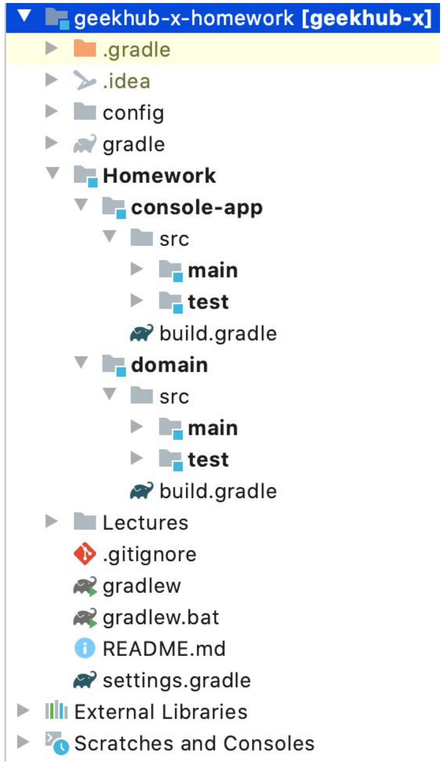
Homework Task 2. DateTime

- Improve the logging mechanism by adding time to the logs in the format "12-31-2021 15:30:59:904"
- Add the "creationDate" field to the Lection object.
- Add sorting of lectures by date ASC/DESC
- Add the "DeadLine" field to the Homework object

Homework Task (Gradle)

- Integrate Gradle into project
- Take full advantage of the gradle's capabilities to pull up libraries
- Migrate adding of third-party libs by using Gradle
- Create two child modules of “Homework” module:
  - domain (store only business logic code)
  - console-app (store console app related code)

Homework Task Example Project Structure



```
▼ geekhub-x-homework [geekhub-x]
    ▶ .gradle
    ▶ .idea
    ▶ config
    ▶ gradle
    ▼ Homework
        ▼ console-app
            ▼ src
                ▶ main
                ▶ test
              build.gradle
        ▼ domain
            ▼ src
                ▶ main
                ▶ test
              build.gradle
    ▶ Lectures
      .gitignore
      gradlew
      gradlew.bat
      README.md
      settings.gradle
▶ External Libraries
▶ Scratches and Consoles
```

Links

https://www.baeldung.com/java-optional

https://www.baeldung.com/java-8-date-time-intro

https://www.baeldung.com/gradle

Thanks!
Any Questions?

Or find us in Slack:

@Vladyslav Nikolenko
@Bogdan Cherniak
@Volodymyr Vedula