hello

CoEs

JAVA OOP ORIENTATION

netcompany

Agenda

- Introduction.
- Programming paradigms.
- OOP 101.
- The case study.
- Example from real projects.
- Summary.
- Practical.

Introduction

Phan Cong Thuc

- A Software Engineer
- More than 6 years of working experience in IT industry.
- Joined Netcompany Vietnam since May 2019 as a Consultant.
- Promoted to Master from January 1, 2023.

Introduction

Nguyen Le Nhat Truong

- B.Sc in Software Engineering by FPT Uni.
- 9 years of working experience in IT industry.
- I started in Netcompany in November 2020 as a consultant:
  - Developer in a Web application project.
  - Team lead and software architect.
- Today I'm the Senior Architect for Netcompany Vietnam.

Netcompany at a glance

Netcompany is one of Europe's fastest growing and most successful IT services companies, leading the way in showing how digital transformation can create strong, sustainable societies, successful companies and better lives for us all.

Our ambition is to become the market leader within IT services in Europe. The acquisition of Intrasoft in 2021 has strengthened our foundation to achieve that ambition - further expanding our portfolio of platforms and unique expertise across sectors with a global headcount of +7,400 talented employees.

By building flexible, scalable and secure digital platforms, Netcompany is positioned to help Europe thrive through a decade of massive digitisation.

Netcompany Core

3.8bn
Revenue (DKK) in 2022

+20%
Avg. revenue growth for +10 years

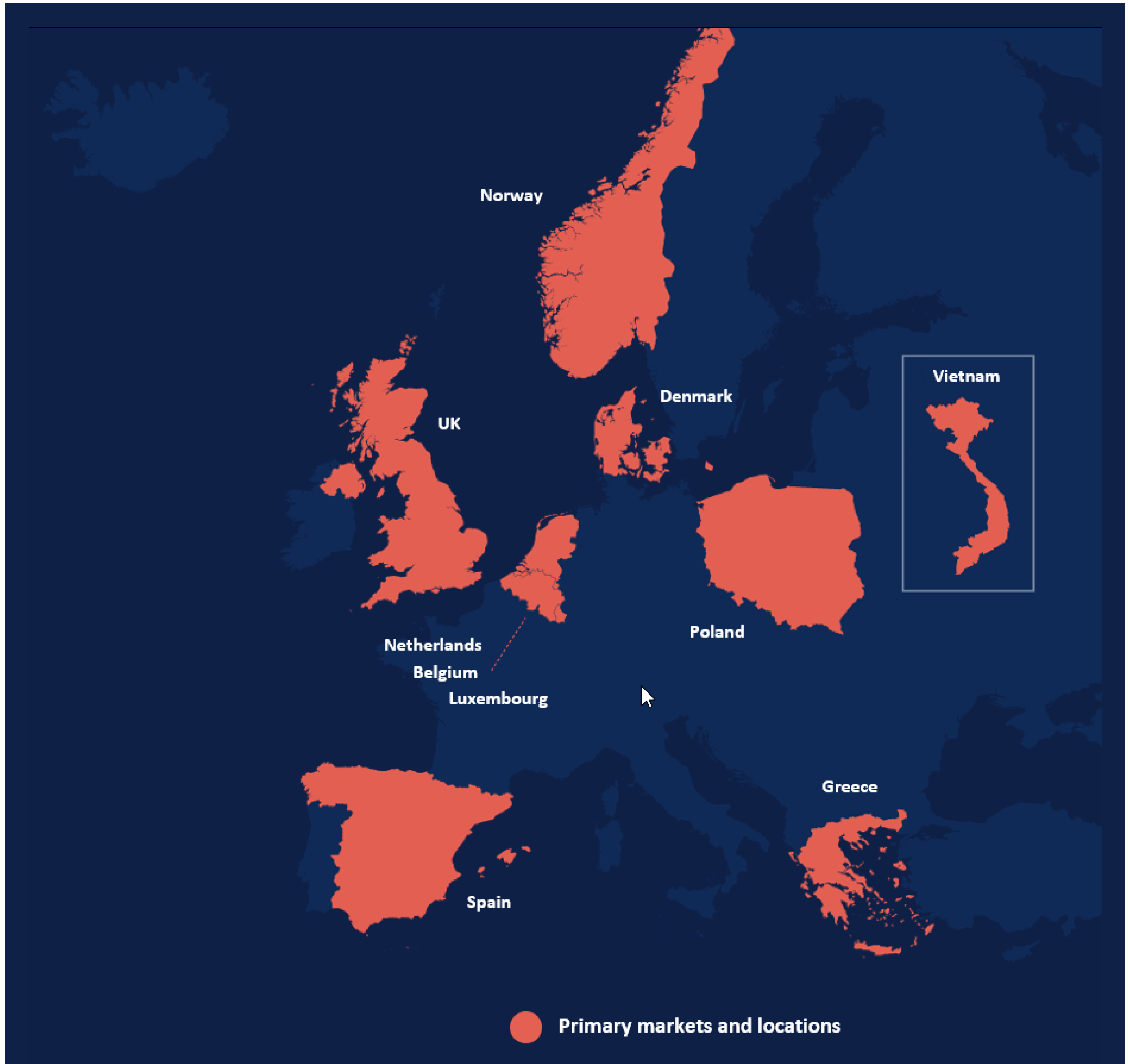
+4,100
Employees year-end 2022

Netcompany-Intrasoft

1.7bn
Revenue (DKK) in 2022

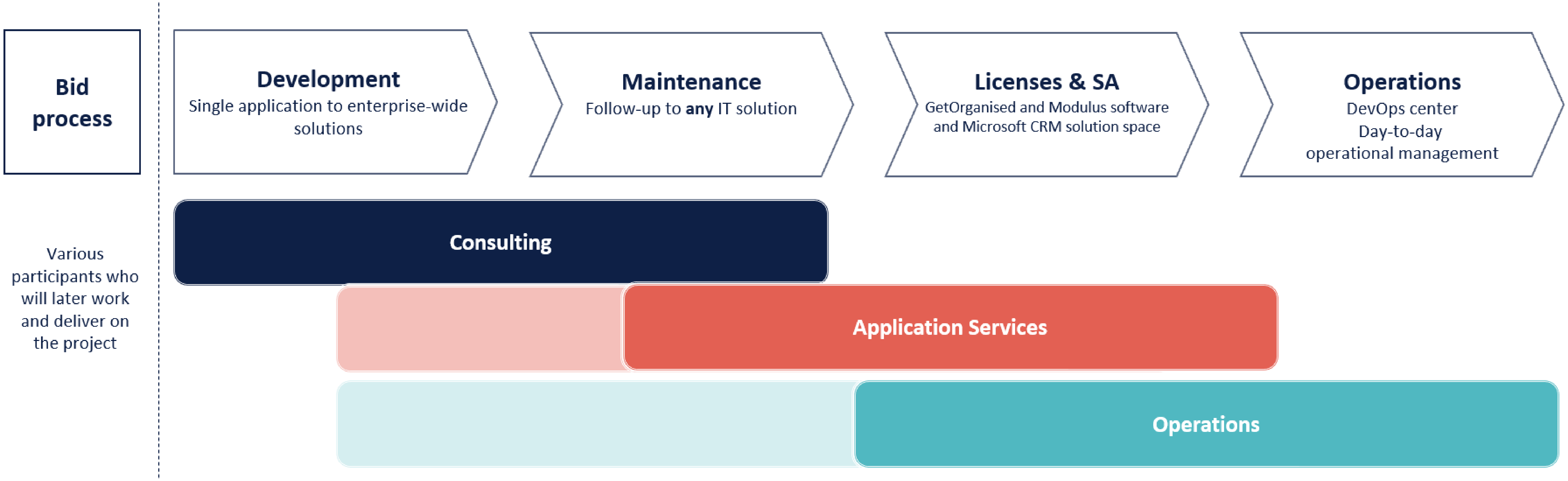
+8%
Avg. revenue growth for +10 years (+12% in 2022)

+3,200
Employees year-end 2022



Additional offices in:
Romania | Cyprus | Jordan | Kenya | South Africa

An active end-to-end service provider



Bid process

Various participants who will later work and deliver on the project

Development
Single application to enterprise-wide solutions

Maintenance
Follow-up to **any** IT solution

Licenses & SA
GetOrganised and Modulus software and Microsoft CRM solution space

Operations
DevOps center
Day-to-day
operational management

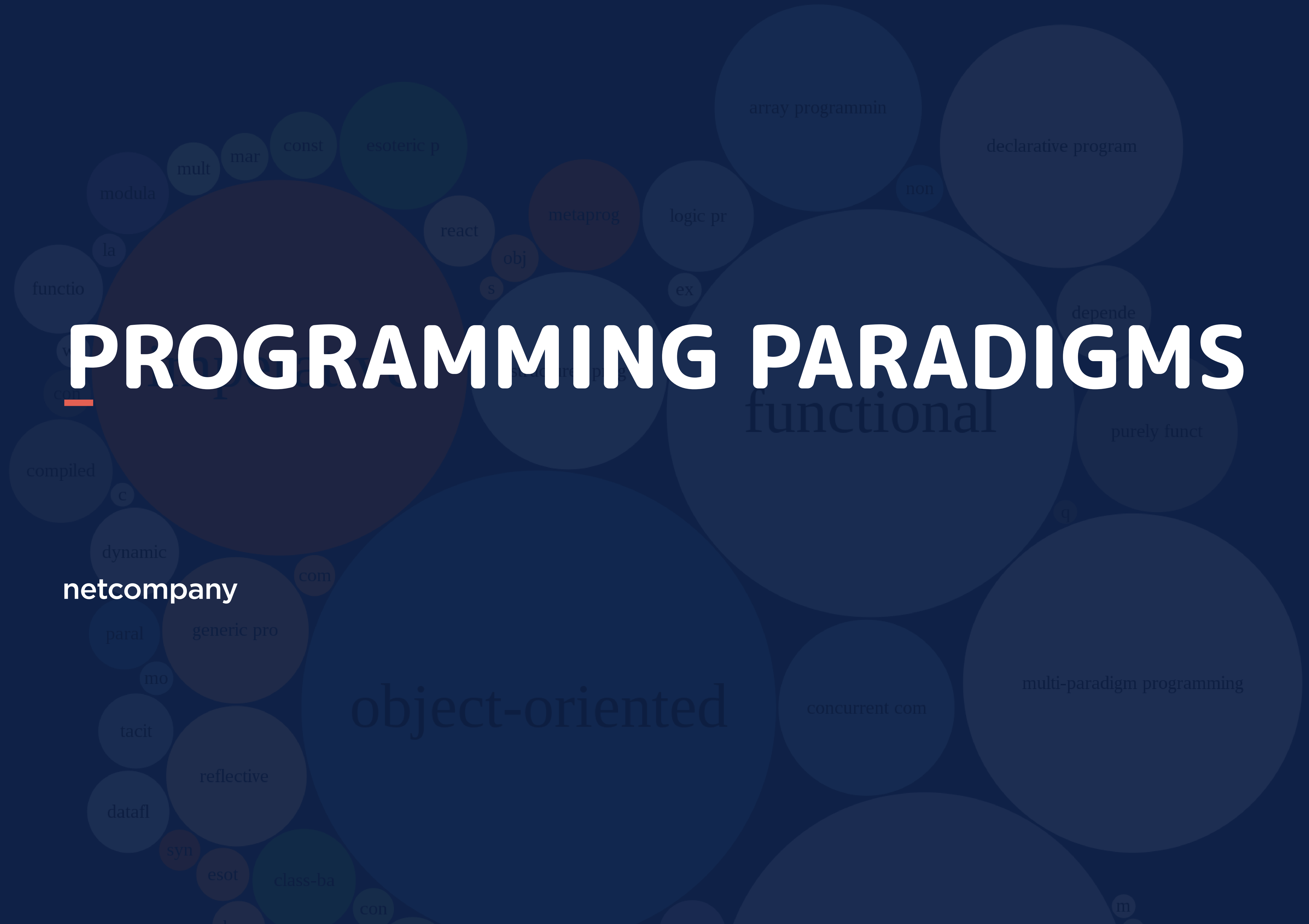
Consulting

Application Services

Operations

PROGRAMMING PARADIGMS

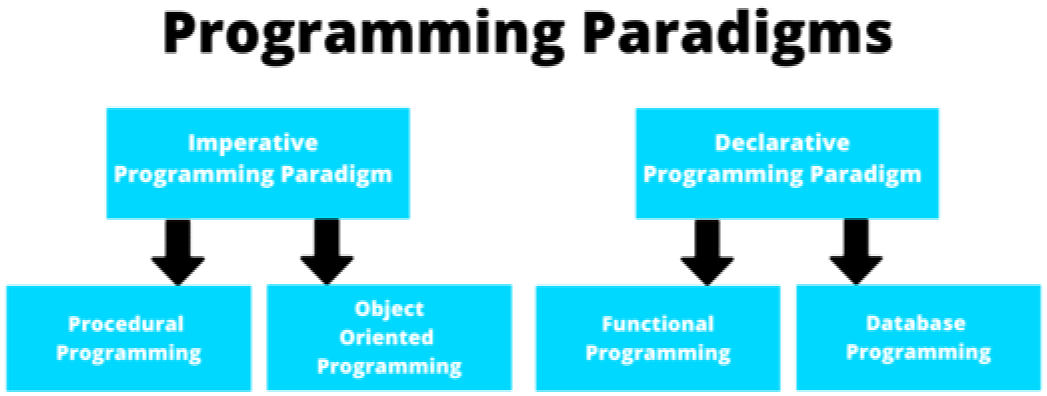
netcompany



Programming paradigms

- Is a way/style/approach of programming in order to solve a given problem

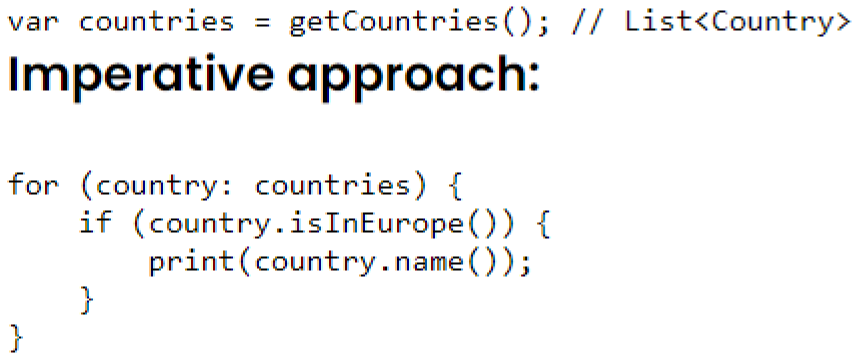
- There are two groups of programming paradigm:


Programming Paradigms
Imperative Programming Paradigm
Declarative Programming Paradigm
Procedural Programming
Object Oriented Programming
Functional Programming
Database Programming

Programming paradigms

Two groups:

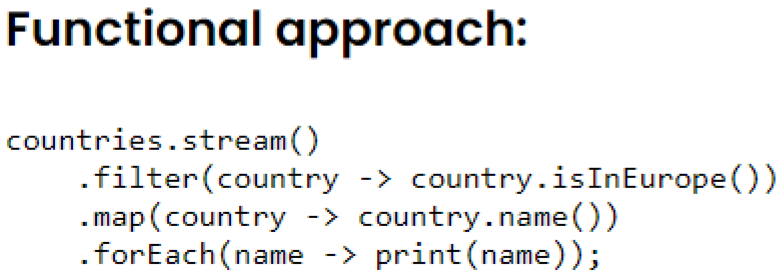
- Imperative: how to achieve the goal, step by step, focus on the flow
- Declarative: what to achieve, focus on the desired result.



```
var countries = getCountries(); // List<Country>
```

Imperative approach:

```
for (country: countries) {
    if (country.isInEurope()) {
        print(country.name());
    }
}
```



Functional approach:

```
countries.stream()
    .filter(country -> country.isInEurope())
    .map(country -> country.name())
    .forEach(name -> print(name));
```

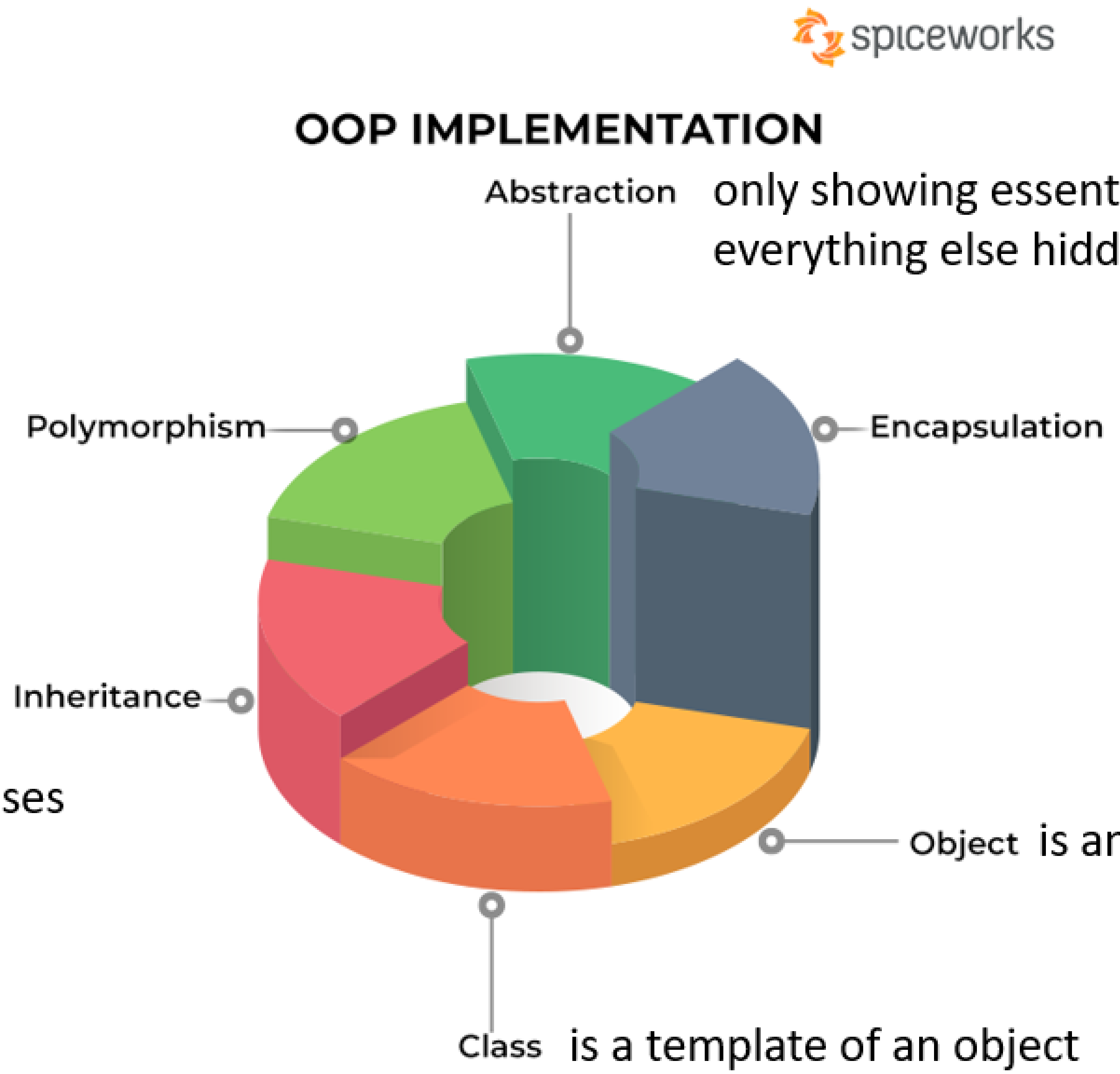
OOP 101

netcompany

OOP 101

> Object-oriented programming (OOP) is a programming paradigm based on the concept of "**objects**", which can contain data and code. The data is in the form of fields (often known as **attributes** or **properties**), and the code is in the form of procedures (often known as **methods**).

OOP 101



only showing essential details and keeping everything else hidden

methods that can take on many forms

- dynamic polymorphism
- static polymorphism

bundling data with methods that can operate on that data within a class

classes can derive from other classes

is an instance of a class

is a template of an object

THE CASE STUDY

netcompany

The case study

Context

Greeting Dazzle the Coder

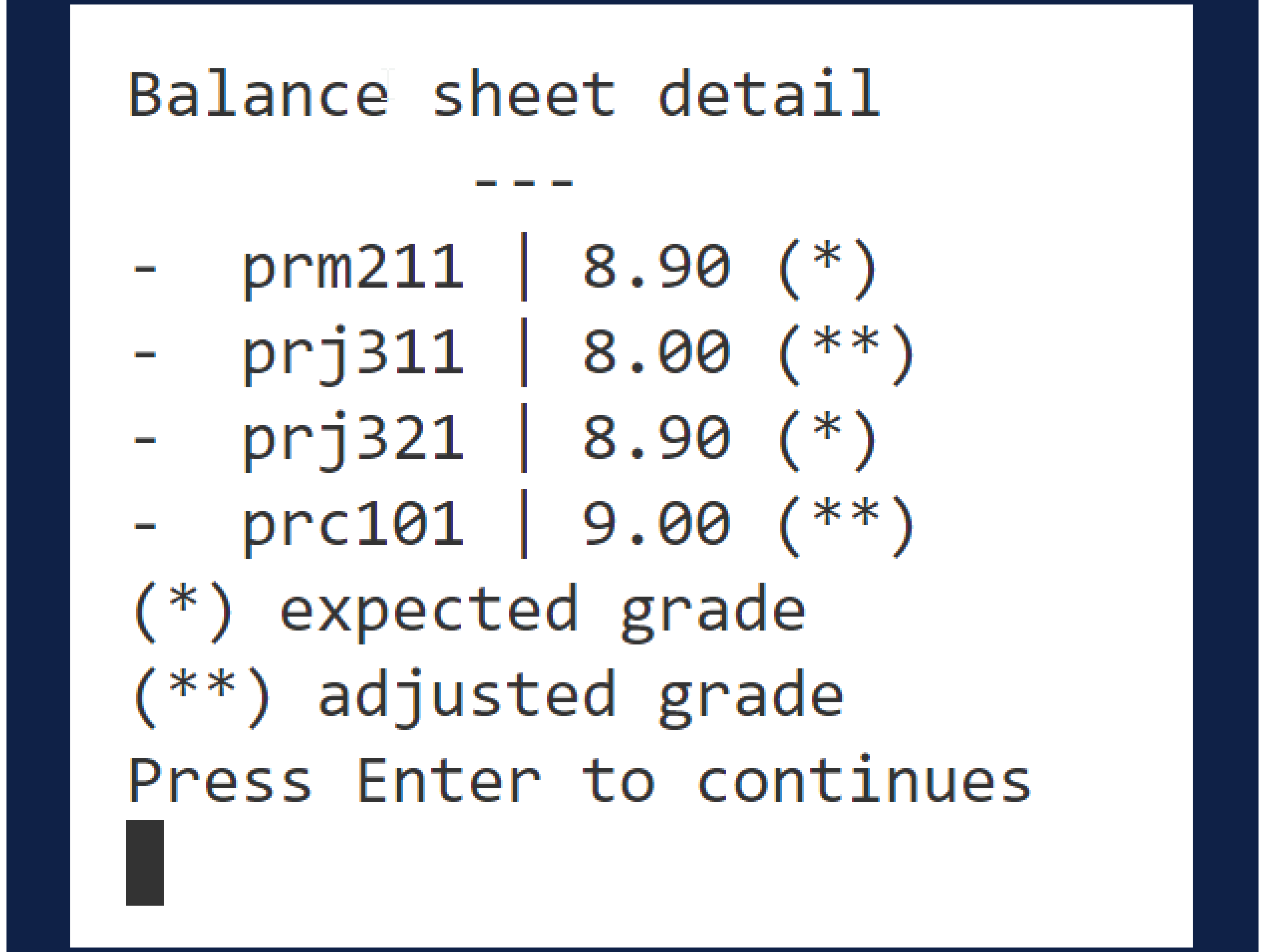
- A junior student who have very good programming skill
- He has a cool project to manage his own GPA
- He just finished his OOP course in the last semester

The case study

Context

Meet his GPA management app:

- A console application writen in JAVA
- Allow student manage their courses
- Allow student to input or clean grade for courses and view them
- Create a GPA balance sheet base on their current GPA and their expected
- Allow student to save manage their adjusted balance sheets
- They can view or clean a saved sheet
- A cool tool to planning for a new semester



```
Balance sheet detail
         ---
-  prm211 | 8.90 (*)
-  prj311 | 8.00 (**)
-  prj321 | 8.90 (*)
-  prc101 | 9.00 (**)
(*) expected grade
(**) adjusted grade
Press Enter to continues
█
```

The case study

Context

Greeting Dazzle's best friend, Huskar

- He is Dazzle's bestie
- He has a huge passion with programming
- His programming skill is not as good as Dazzle
- He is a loyal user of Dazzle's App
- He shared a lot of ideas about this App and wants to help Dazzle to build up this App

The case study

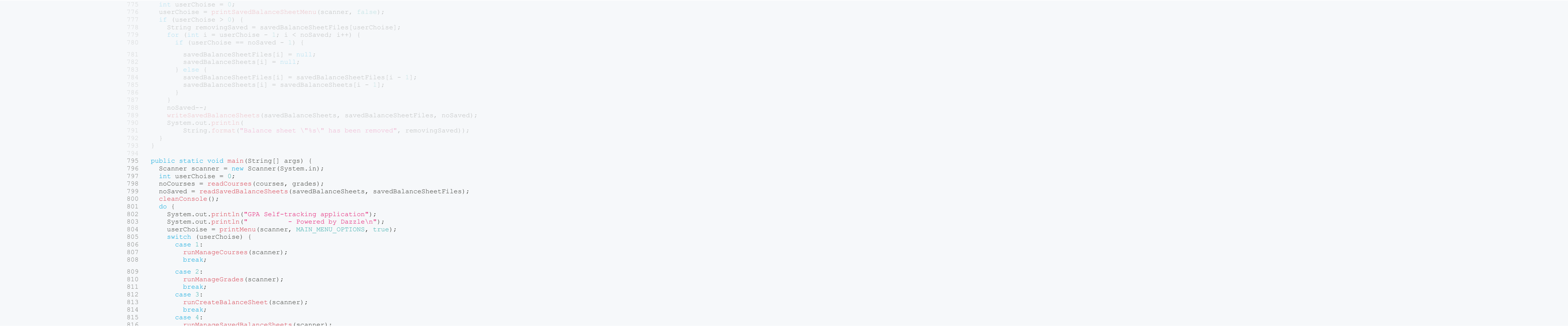
Context

In the very first day, they facing some collaboration problems

- It's take Dazzle a while to onboard Huskar to the project

The case study

Context



```java
  int userChoise = 0;
  userChoise = printSavedBalanceSheetMenu(scanner, false);
  if (userChoise > 0) {
    String removingSaved = savedBalanceSheetFiles[userChoise];
    for (int i = userChoise - 1; i < noSaved; i++) {
      if (userChoise == noSaved - 1) {
        savedBalanceSheetFiles[i] = null;
        savedBalanceSheets[i] = null;
      } else {
        savedBalanceSheetFiles[i] = savedBalanceSheetFiles[i - 1];
        savedBalanceSheets[i] = savedBalanceSheets[i - 1];
      }
    }
    noSaved--;
    writeSavedBalanceSheets(savedBalanceSheets, savedBalanceSheetFiles, noSaved);
    System.out.println(
        String.format("Balance sheet \"%s\" has been removed", removingSaved));
  }
}

public static void main(String[] args) {
  Scanner scanner = new Scanner(System.in);
  int userChoise = 0;
  noCourses = readCourses(courses, grades);
  noSaved = readSavedBalanceSheets(savedBalanceSheets, savedBalanceSheetFiles);
  cleanConsole();
  do {
    System.out.println("GPA Self-tracking application");
    System.out.println("           - Powered by Dazzle\n");
    userChoise = printMenu(scanner, MAIN_MENU_OPTIONS, true);
    switch (userChoise) {
      case 1:
        runManageCourses(scanner);
        break;
      case 2:
        runManageGrades(scanner);
        break;
      case 3:
        runCreateBalanceSheet(scanner);
        break;
      case 4:
        runManageSavedBalanceSheets(scanner);
```

The case study

Context

In the very first day, they facing some collaboration problems

- It's take Dazzle a while to onboard Huskar to the project
- There is major file conflict for every new feature they develop together
- Duplicated code everywhere
- **Take time to develop features concurrently**

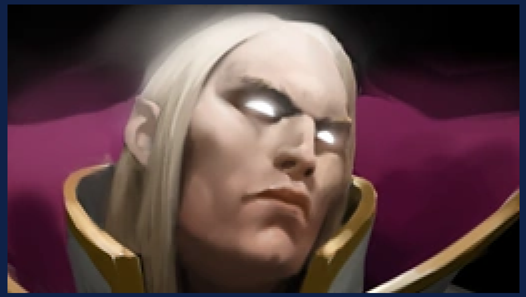
The case study

Context

Dazzle and Huskar try to get help from a senior student, Invoker. He's give them some advises:

- Re-organize their code so the human can read easyly.
- Using OOP as a fundamental tool to refactor their project.
- Conduct a human friendly code structure and arrange classes into it.

Let follow their journey to build-up a maintainable project.



The case study

Abstraction - The foundation of the OOP

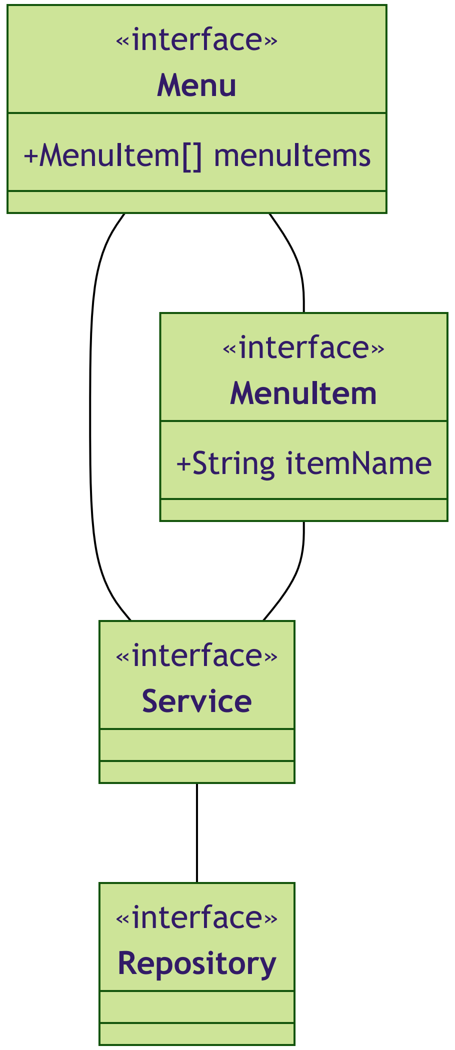
- Seperate WHAT from HOW
- Hide the complex behide the Abstraction

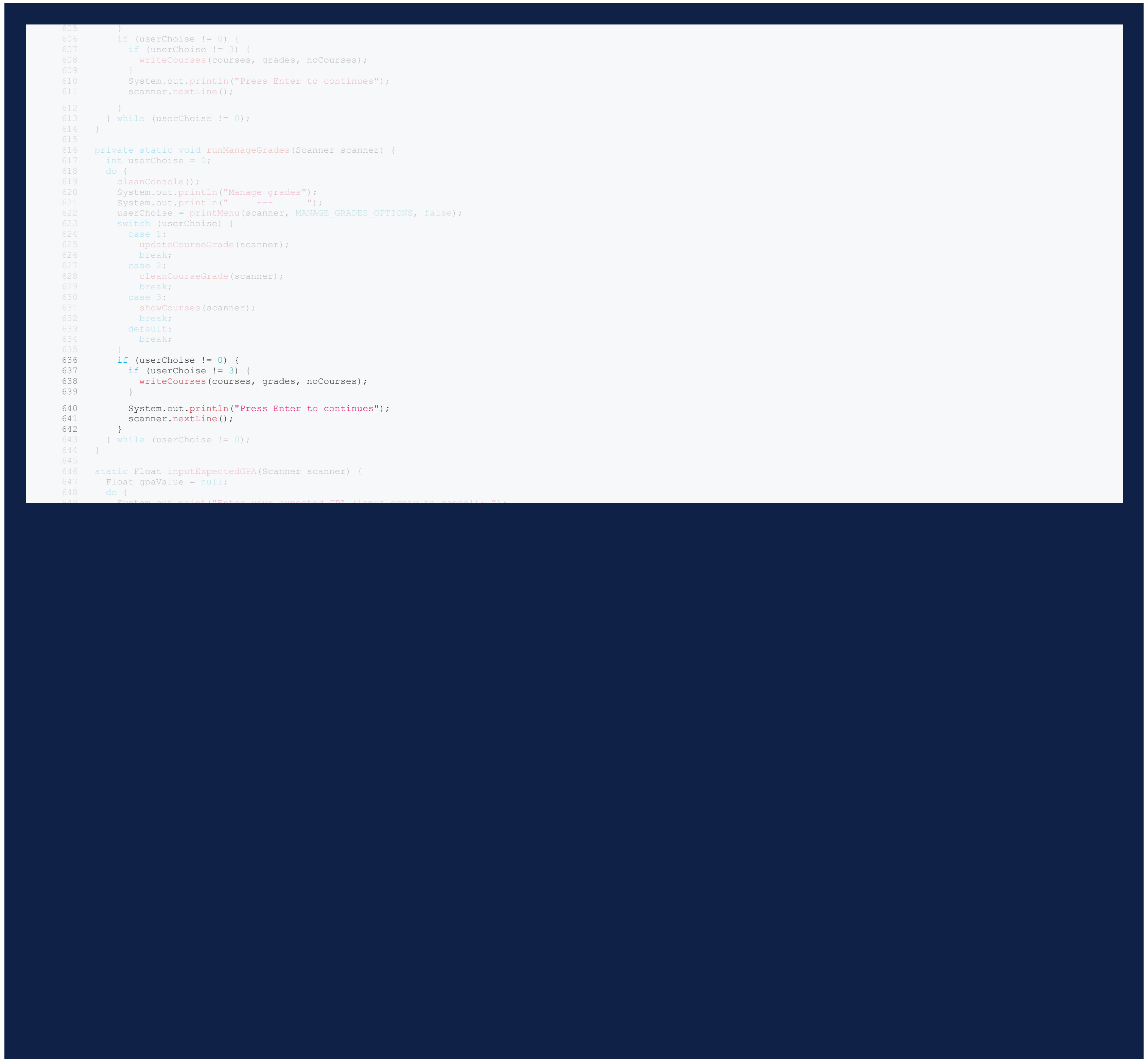
- Focus on high-level idea, seperately from the low-level details

The case study

Abstraction - The foundation of the OOP

- Seperate WHAT from HOW



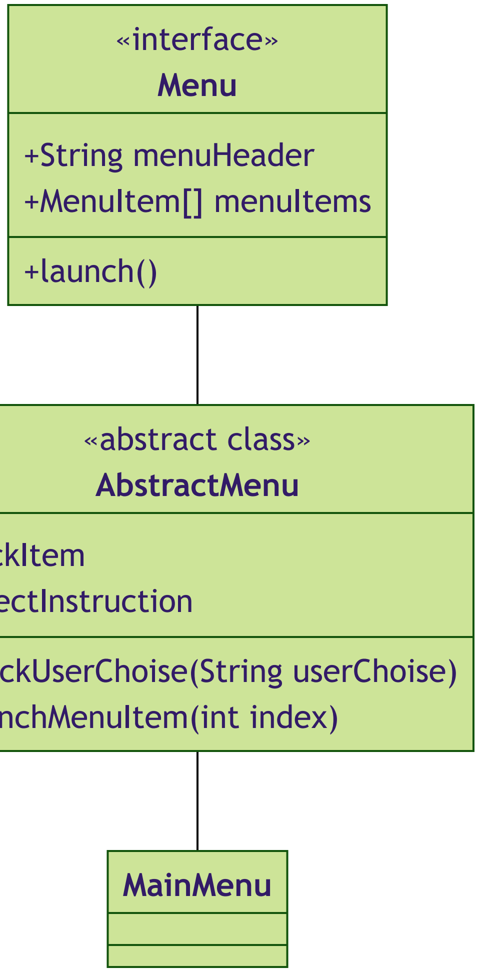


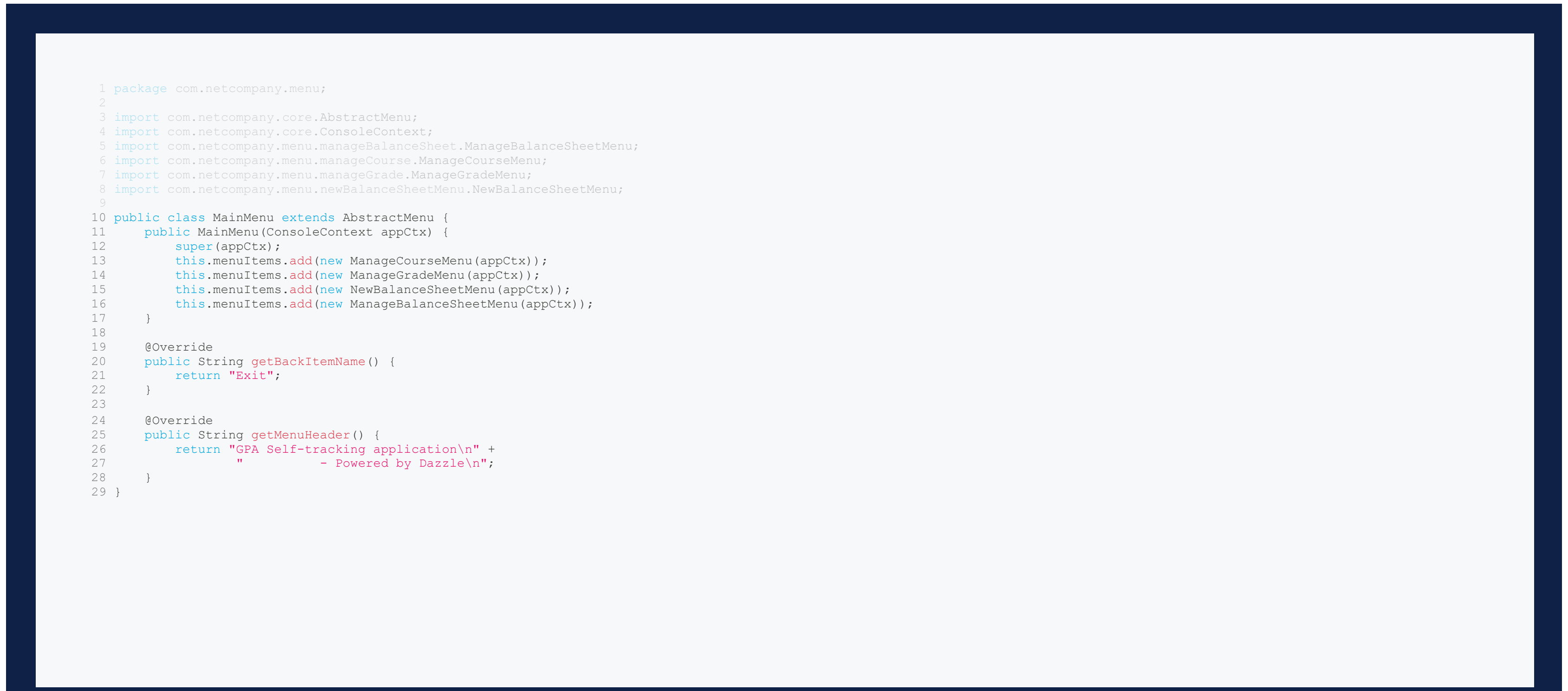
```
      }
      if (userChoise != 0) {
        if (userChoise != 3) {
          writeCourses(courses, grades, noCourses);
        }
        System.out.println("Press Enter to continues");
        scanner.nextLine();

      }
    } while (userChoise != 0);
  }

  private static void runManageGrades(Scanner scanner) {
    int userChoise = 0;
    do {
      cleanConsole();
      System.out.println("Manage grades");
      System.out.println("     ---       ");
      userChoise = printMenu(scanner, MANAGE_GRADES_OPTIONS, false);
      switch (userChoise) {
        case 1:
          updateCourseGrade(scanner);
          break;
        case 2:
          cleanCourseGrade(scanner);
          break;
        case 3:
          showCourses(scanner);
          break;
        default:
          break;
      }
      if (userChoise != 0) {
        if (userChoise != 3) {
          writeCourses(courses, grades, noCourses);
        }

        System.out.println("Press Enter to continues");
        scanner.nextLine();
      }
    } while (userChoise != 0);
  }

  static Float inputExpectedGPA(Scanner scanner) {
    Float gpaValue = null;
    do {
      System.out.print("Enter your expected GPA (input empty to cancel): ");
```

The case study

Abstraction - The foundation of the OOP

- Hide the complex behide the Abstraction



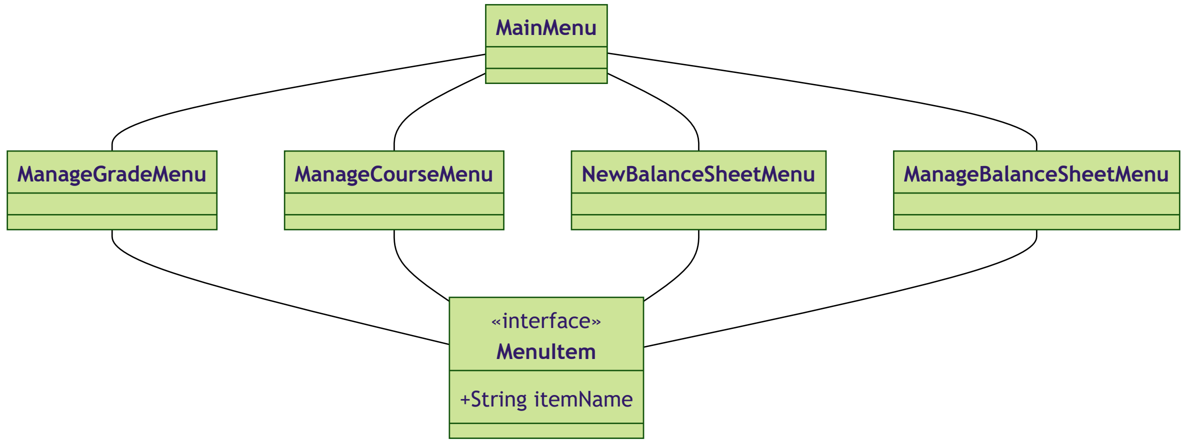


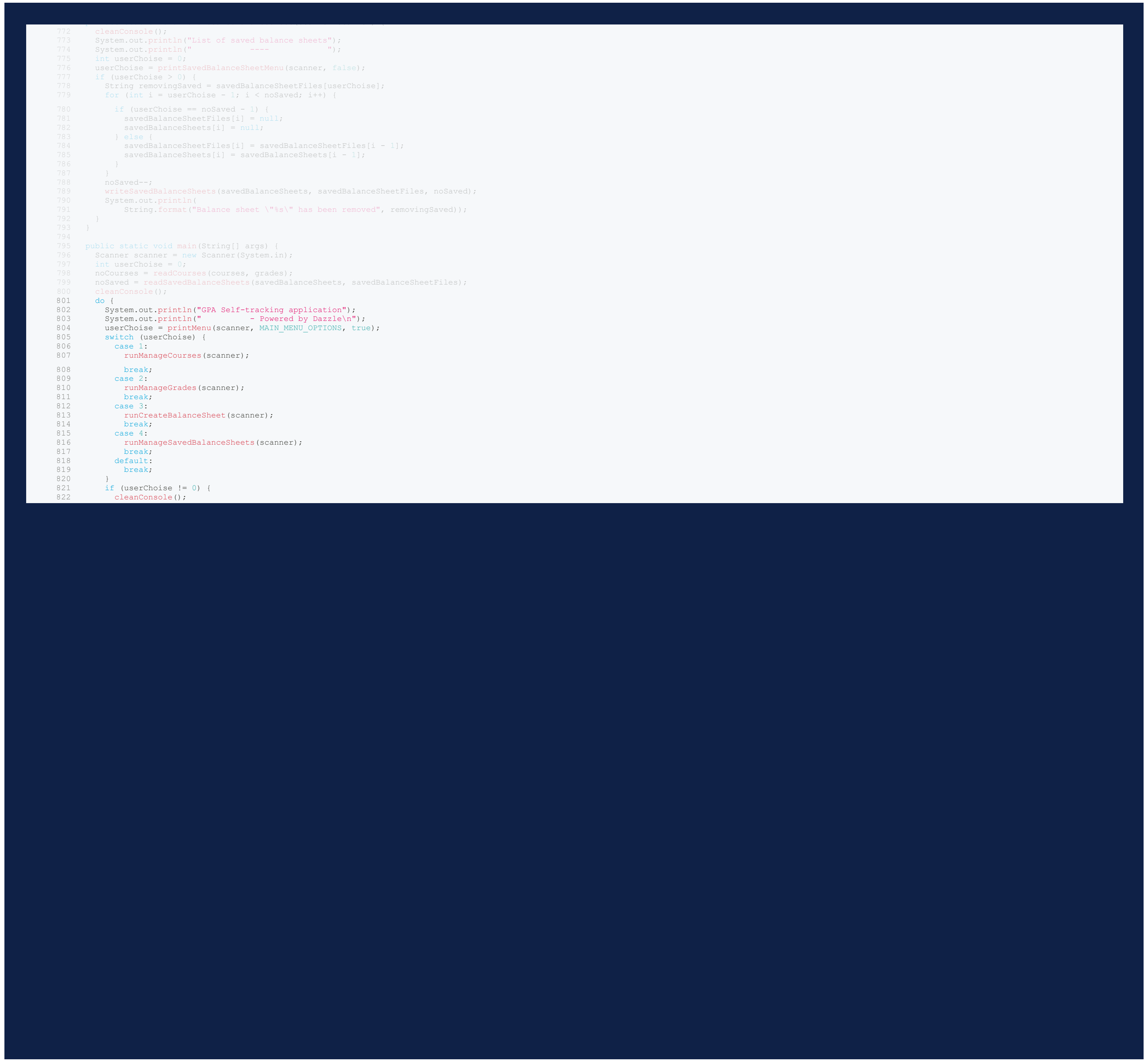
```java
package com.netcompany.menu;

import com.netcompany.core.AbstractMenu;
import com.netcompany.core.ConsoleContext;
import com.netcompany.menu.manageBalanceSheet.ManageBalanceSheetMenu;
import com.netcompany.menu.manageCourse.ManageCourseMenu;
import com.netcompany.menu.manageGrade.ManageGradeMenu;
import com.netcompany.menu.newBalanceSheetMenu.NewBalanceSheetMenu;

public class MainMenu extends AbstractMenu {
    public MainMenu(ConsoleContext appCtx) {
        super(appCtx);
        this.menuItems.add(new ManageCourseMenu(appCtx));
        this.menuItems.add(new ManageGradeMenu(appCtx));
        this.menuItems.add(new NewBalanceSheetMenu(appCtx));
        this.menuItems.add(new ManageBalanceSheetMenu(appCtx));
    }

    @Override
    public String getBackItemName() {
        return "Exit";
    }

    @Override
    public String getMenuHeader() {
        return "GPA Self-tracking application\n" +
                "          - Powered by Dazzle\n";
    }
}
```

The case study

Abstraction - The foundation of the OOP

- Focus on high-level idea, seperately from the low-level details



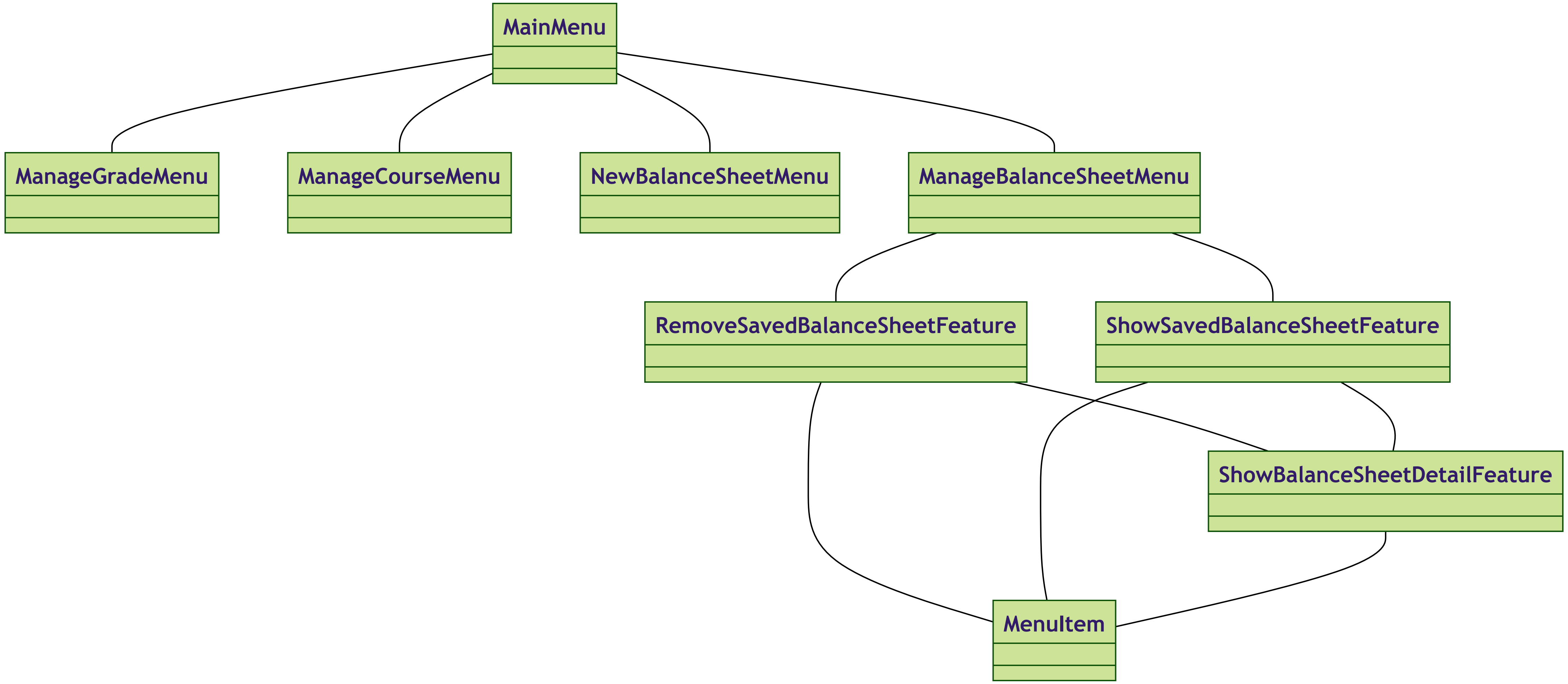


```java
    cleanConsole();
    System.out.println("List of saved balance sheets");
    System.out.println("               ----              ");
    int userChoise = 0;
    userChoise = printSavedBalanceSheetMenu(scanner, false);
    if (userChoise > 0) {
      String removingSaved = savedBalanceSheetFiles[userChoise];
      for (int i = userChoise - 1; i < noSaved; i++) {
        if (userChoise == noSaved - 1) {
          savedBalanceSheetFiles[i] = null;
          savedBalanceSheets[i] = null;
        } else {
          savedBalanceSheetFiles[i] = savedBalanceSheetFiles[i - 1];
          savedBalanceSheets[i] = savedBalanceSheets[i - 1];
        }
      }
      noSaved--;
      writeSavedBalanceSheets(savedBalanceSheets, savedBalanceSheetFiles, noSaved);
      System.out.println(
          String.format("Balance sheet \"%s\" has been removed", removingSaved));
    }
  }

  public static void main(String[] args) {
    Scanner scanner = new Scanner(System.in);
    int userChoise = 0;
    noCourses = readCourses(courses, grades);
    noSaved = readSavedBalanceSheets(savedBalanceSheets, savedBalanceSheetFiles);
    cleanConsole();
    do {
      System.out.println("GPA Self-tracking application");
      System.out.println("          - Powered by Dazzle\n");
      userChoise = printMenu(scanner, MAIN_MENU_OPTIONS, true);
      switch (userChoise) {
        case 1:
          runManageCourses(scanner);

          break;
        case 2:
          runManageGrades(scanner);
          break;
        case 3:
          runCreateBalanceSheet(scanner);
          break;
        case 4:
          runManageSavedBalanceSheets(scanner);
          break;
        default:
          break;
      }
      if (userChoise != 0) {
        cleanConsole();
```

The case study

Abstraction - The foundation of the OOP

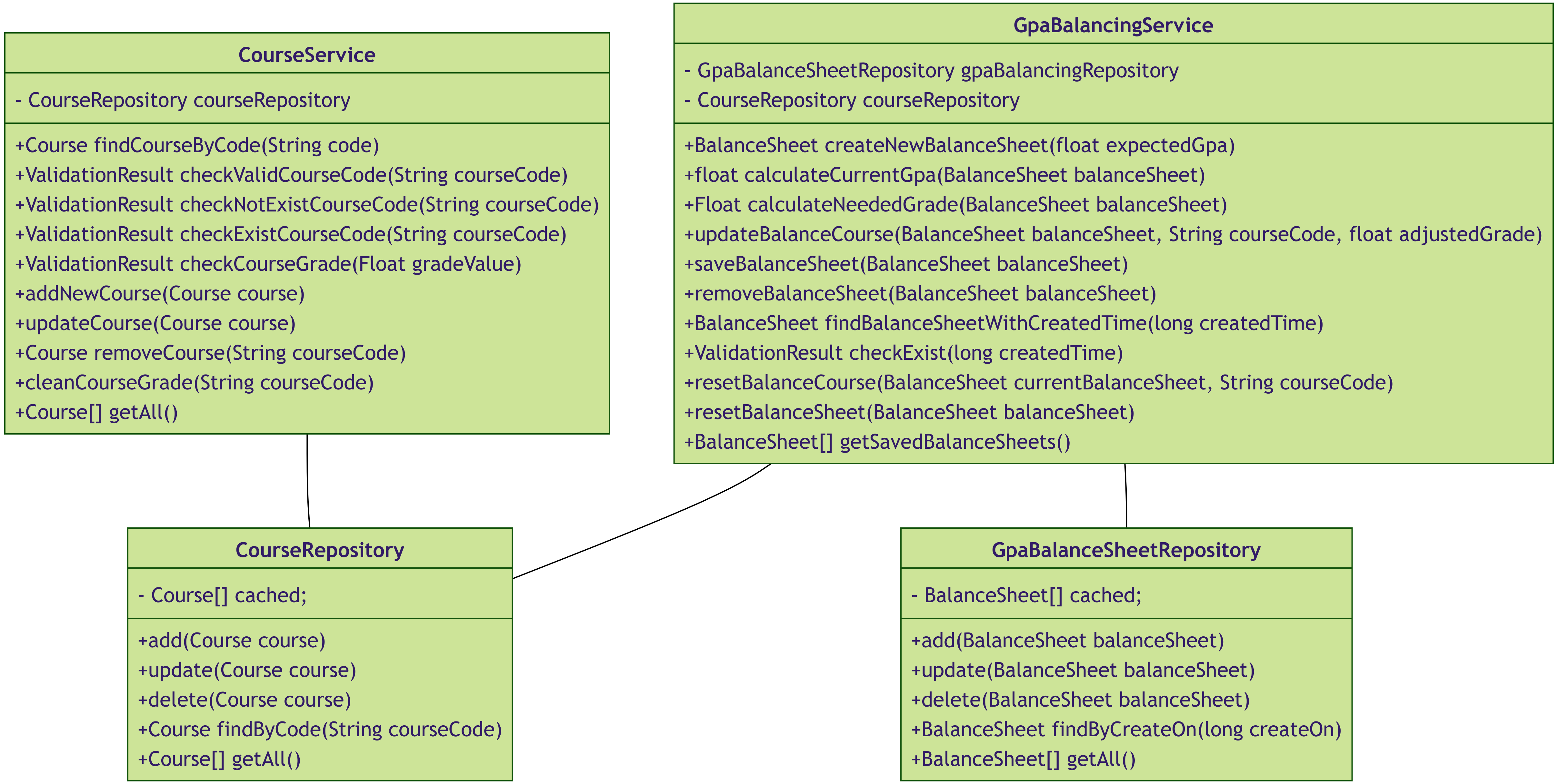
- Focus on high-level idea, seperately from the low-level details



The case study

Abstraction - The foundation of the OOP

- Focus on high-level idea, seperately from the low-level details



CourseService

- CourseRepository courseRepository

+Course findCourseByCode(String code)
+ValidationResult checkValidCourseCode(String courseCode)
+ValidationResult checkNotExistCourseCode(String courseCode)
+ValidationResult checkExistCourseCode(String courseCode)
+ValidationResult checkCourseGrade(Float gradeValue)
+addNewCourse(Course course)
+updateCourse(Course course)
+Course removeCourse(String courseCode)
+cleanCourseGrade(String courseCode)
+Course[] getAll()

GpaBalancingService

- GpaBalanceSheetRepository gpaBalancingRepository
- CourseRepository courseRepository

+BalanceSheet createNewBalanceSheet(float expectedGpa)
+float calculateCurrentGpa(BalanceSheet balanceSheet)
+Float calculateNeededGrade(BalanceSheet balanceSheet)
+updateBalanceCourse(BalanceSheet balanceSheet, String courseCode, float adjustedGrade)
+saveBalanceSheet(BalanceSheet balanceSheet)
+removeBalanceSheet(BalanceSheet balanceSheet)
+BalanceSheet findBalanceSheetWithCreatedTime(long createdTime)
+ValidationResult checkExist(long createdTime)
+resetBalanceCourse(BalanceSheet currentBalanceSheet, String courseCode)
+resetBalanceSheet(BalanceSheet balanceSheet)
+BalanceSheet[] getSavedBalanceSheets()

CourseRepository

- Course[] cached;

+add(Course course)
+update(Course course)
+delete(Course course)
+Course findByCode(String courseCode)
+Course[] getAll()

GpaBalanceSheetRepository

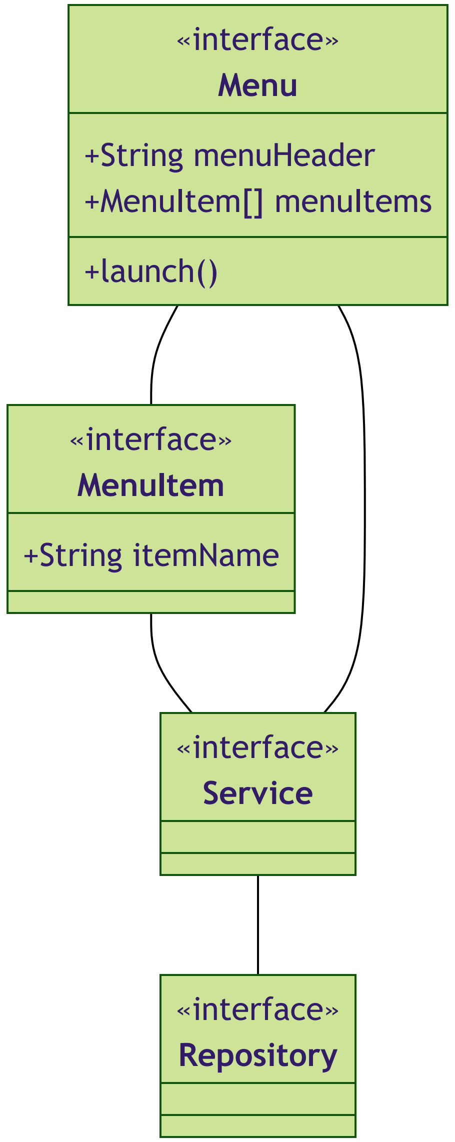
- BalanceSheet[] cached;

+add(BalanceSheet balanceSheet)
+update(BalanceSheet balanceSheet)
+delete(BalanceSheet balanceSheet)
+BalanceSheet findByCreateOn(long createOn)
+BalanceSheet[] getAll()

The case study

Abstraction - The foundation of the OOP

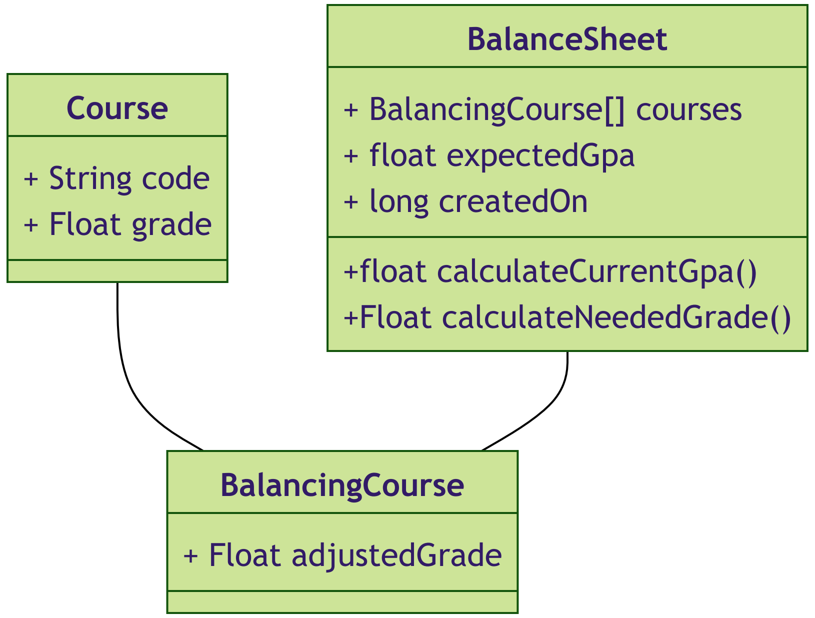
- Seperate WHAT from HOW
- Hide the complex behide the Abstraction
- Focus on high-level idea, seperately from the low-level details



The case study

Abstraction

- It's not just about the domain objects

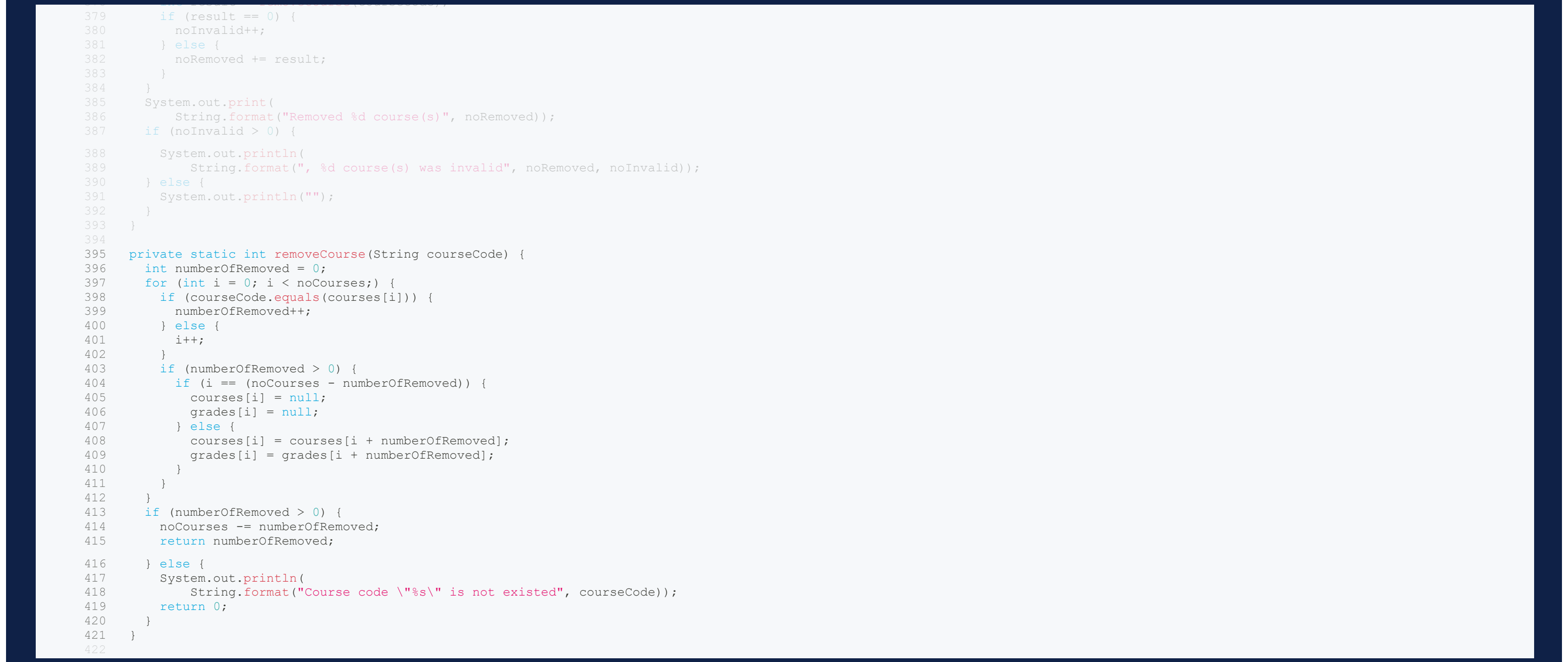


The case study

Encapsulation - The state management for the Objects

The current project overview:

- Sharing the global variables between functions
- Easy to access and build new functions on it
- Expose the data integrity to all functions without business logic checking



```java
    if (result == 0) {
      noInvalid++;
    } else {
      noRemoved += result;
    }
  }
  System.out.print(
      String.format("Removed %d course(s)", noRemoved));
  if (noInvalid > 0) {
    System.out.println(
        String.format(", %d course(s) was invalid", noRemoved, noInvalid));
  } else {
    System.out.println("");
  }
}

private static int removeCourse(String courseCode) {
  int numberOfRemoved = 0;
  for (int i = 0; i < noCourses;) {
    if (courseCode.equals(courses[i])) {
      numberOfRemoved++;
    } else {
      i++;
    }
    if (numberOfRemoved > 0) {
      if (i == (noCourses - numberOfRemoved)) {
        courses[i] = null;
        grades[i] = null;
      } else {
        courses[i] = courses[i + numberOfRemoved];
        grades[i] = grades[i + numberOfRemoved];
      }
    }
  }
  if (numberOfRemoved > 0) {
    noCourses -= numberOfRemoved;
    return numberOfRemoved;
  } else {
    System.out.println(
        String.format("Course code \"%s\" is not existed", courseCode));
    return 0;
  }
}
```

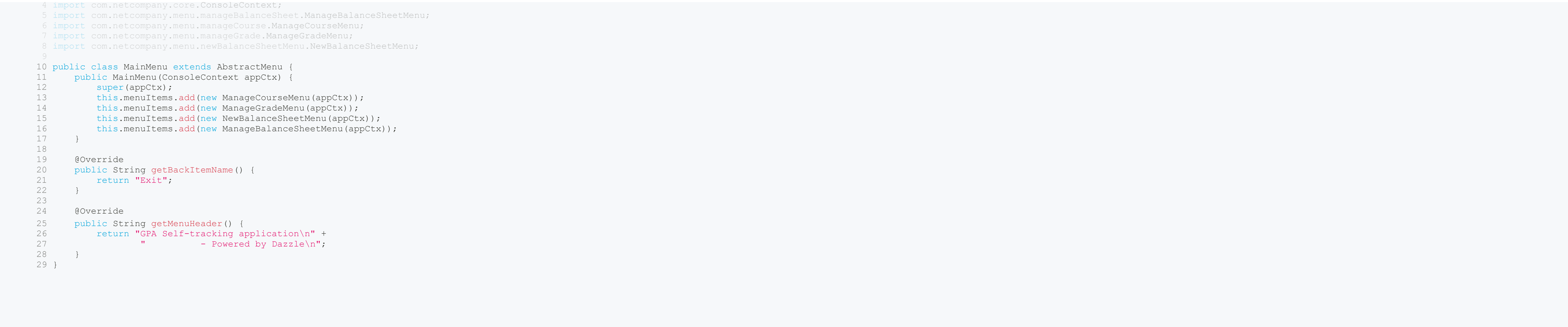
The case study

Inheritance - DRY

- allows to derive a class from another class
- it is a relationship between a superclass and a subclass where subclasses inherits non-private data and behavior from the superclass.
- help to achieve run time polymorphism

The case study

Inheritance - DRY



```java
import com.netcompany.core.ConsoleContext;
import com.netcompany.menu.manageBalanceSheet.ManageBalanceSheetMenu;
import com.netcompany.menu.manageCourse.ManageCourseMenu;
import com.netcompany.menu.manageGrade.ManageGradeMenu;
import com.netcompany.menu.newBalanceSheetMenu.NewBalanceSheetMenu;

public class MainMenu extends AbstractMenu {
    public MainMenu(ConsoleContext appCtx) {
        super(appCtx);
        this.menuItems.add(new ManageCourseMenu(appCtx));
        this.menuItems.add(new ManageGradeMenu(appCtx));
        this.menuItems.add(new NewBalanceSheetMenu(appCtx));
        this.menuItems.add(new ManageBalanceSheetMenu(appCtx));
    }

    @Override
    public String getBackItemName() {
        return "Exit";
    }

    @Override
    public String getMenuHeader() {
        return "GPA Self-tracking application\n" +
                "          - Powered by Dazzle\n";
    }
}
```

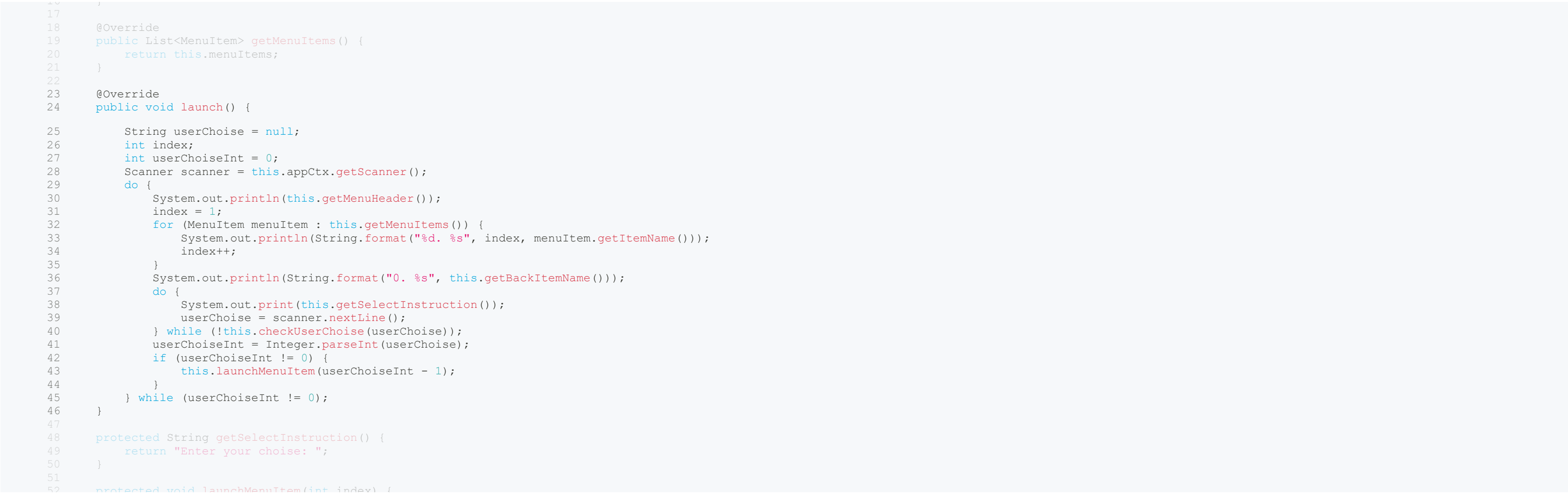
The case study

Polymorphism - The next level of Abstraction

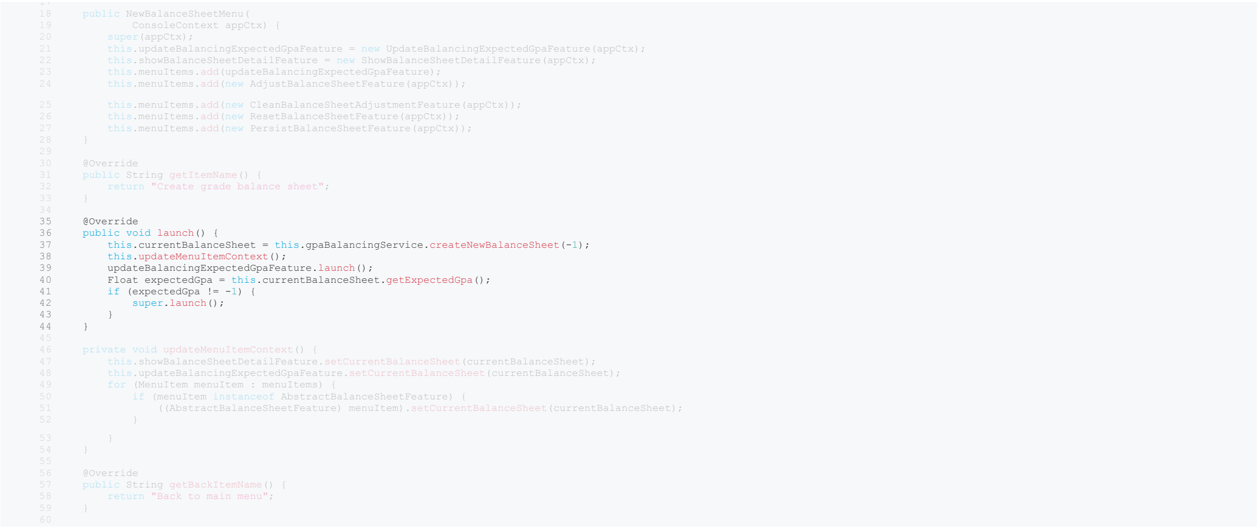
- The word itself indicates the meaning as **poly** means *many* and **morphism** means *types*.
- Polymorphism is a property through which any message can be sent to objects of multiple classes, and every object has the tendency to respond in an appropriate way depending on the class properties.
- This means that polymorphism is the method in an OOP language that does different things depending on the class of the object which calls it.

The case study

Polymorphism



```java
    }

    @Override
    public List<MenuItem> getMenuItems() {
        return this.menuItems;
    }

    @Override
    public void launch() {

        String userChoise = null;
        int index;
        int userChoiseInt = 0;
        Scanner scanner = this.appCtx.getScanner();
        do {
            System.out.println(this.getMenuHeader());
            index = 1;
            for (MenuItem menuItem : this.getMenuItems()) {
                System.out.println(String.format("%d. %s", index, menuItem.getItemName()));
                index++;
            }
            System.out.println(String.format("0. %s", this.getBackItemName()));
            do {
                System.out.print(this.getSelectInstruction());
                userChoise = scanner.nextLine();
            } while (!this.checkUserChoise(userChoise));
            userChoiseInt = Integer.parseInt(userChoise);
            if (userChoiseInt != 0) {
                this.launchMenuItem(userChoiseInt - 1);
            }
        } while (userChoiseInt != 0);
    }

    protected String getSelectInstruction() {
        return "Enter your choise: ";
    }

    protected void launchMenuItem(int index) {
```

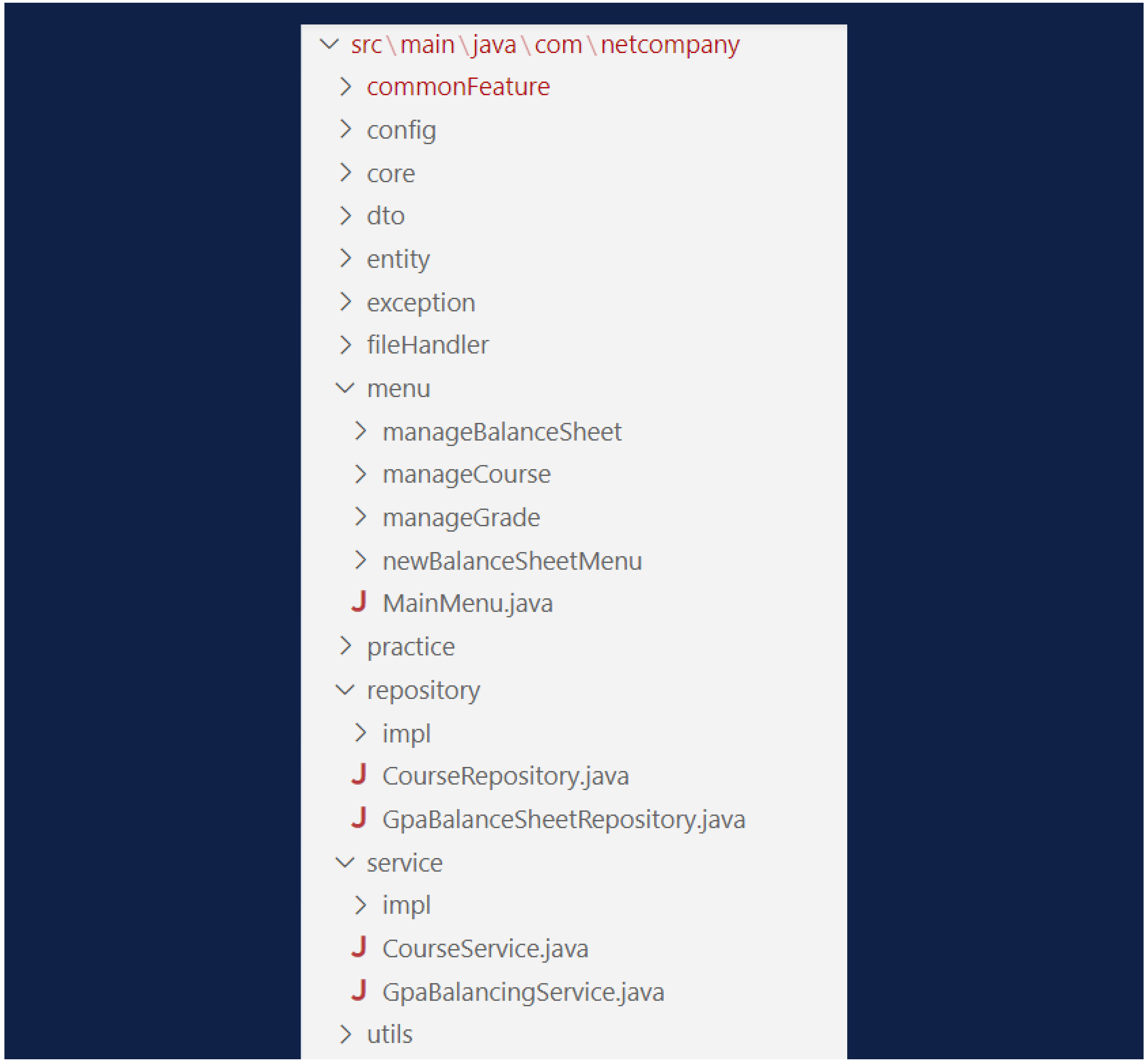


```java
    public NewBalanceSheetMenu(
            ConsoleContext appCtx) {
        super(appCtx);
        this.updateBalancingExpectedGpaFeature = new UpdateBalancingExpectedGpaFeature(appCtx);
        this.showBalanceSheetDetailFeature = new ShowBalanceSheetDetailFeature(appCtx);
        this.menuItems.add(updateBalancingExpectedGpaFeature);
        this.menuItems.add(new AdjustBalanceSheetFeature(appCtx));

        this.menuItems.add(new CleanBalanceSheetAdjustmentFeature(appCtx));
        this.menuItems.add(new ResetBalanceSheetFeature(appCtx));
        this.menuItems.add(new PersistBalanceSheetFeature(appCtx));
    }

    @Override
    public String getItemName() {
        return "Create grade balance sheet";
    }

    @Override
    public void launch() {
        this.currentBalanceSheet = this.gpaBalancingService.createNewBalanceSheet(-1);
        this.updateMenuItemContext();
        updateBalancingExpectedGpaFeature.launch();
        Float expectedGpa = this.currentBalanceSheet.getExpectedGpa();
        if (expectedGpa != -1) {
            super.launch();
        }
    }

    private void updateMenuItemContext() {
        this.showBalanceSheetDetailFeature.setCurrentBalanceSheet(currentBalanceSheet);
        this.updateBalancingExpectedGpaFeature.setCurrentBalanceSheet(currentBalanceSheet);
        for (MenuItem menuItem : menuItems) {
            if (menuItem instanceof AbstractBalanceSheetFeature) {
                ((AbstractBalanceSheetFeature) menuItem).setCurrentBalanceSheet(currentBalanceSheet);
            }

        }
    }

    @Override
    public String getBackItemName() {
        return "Back to main menu";
    }

```

The case study

Conclusion

- Well-organized project
- Easier to locate the issue

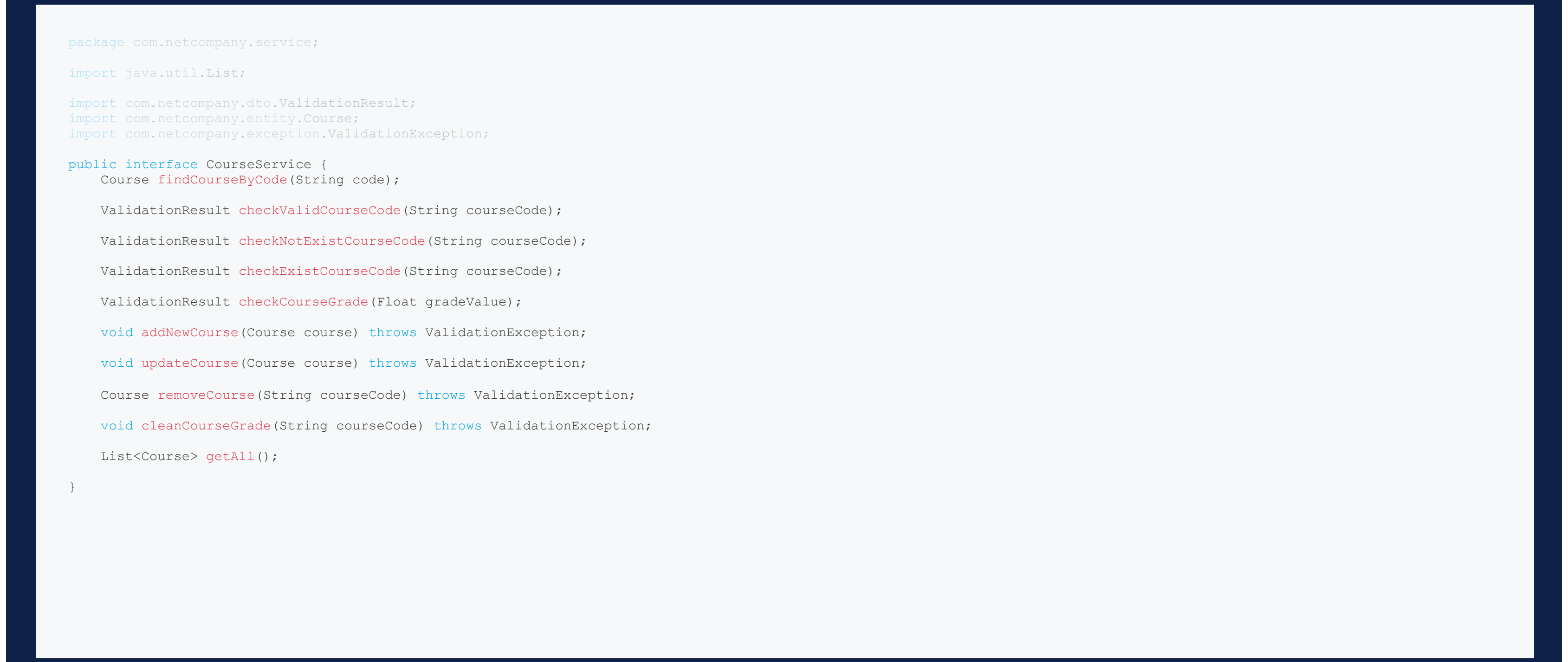


```
src\main\java\com\netcompany
  commonFeature
  config
  core
  dto
  entity
  exception
  fileHandler
  menu
    manageBalanceSheet
    manageCourse
    manageGrade
    newBalanceSheetMenu
    MainMenu.java
  practice
  repository
    impl
    CourseRepository.java
    GpaBalanceSheetRepository.java
  service
    impl
    CourseService.java
    GpaBalancingService.java
  utils
```

The case study

Conclusion

- Well-organized project
- Easier to locate the issue
- Easier to comprehend and start coding

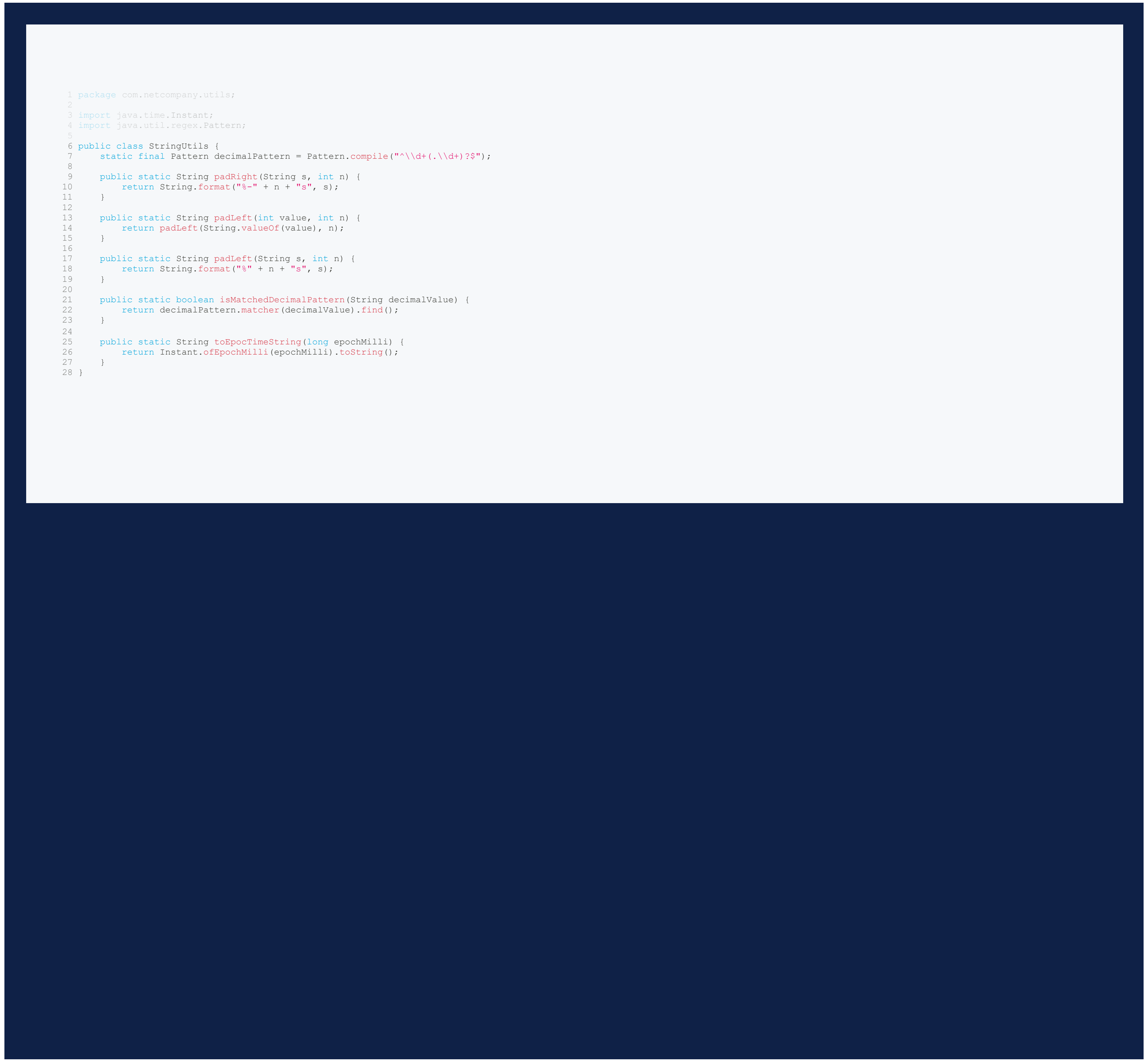


```java
package com.netcompany.service;

import java.util.List;

import com.netcompany.dto.ValidationResult;
import com.netcompany.entity.Course;
import com.netcompany.exception.ValidationException;

public interface CourseService {
    Course findCourseByCode(String code);

    ValidationResult checkValidCourseCode(String courseCode);

    ValidationResult checkNotExistCourseCode(String courseCode);

    ValidationResult checkExistCourseCode(String courseCode);

    ValidationResult checkCourseGrade(Float gradeValue);

    void addNewCourse(Course course) throws ValidationException;

    void updateCourse(Course course) throws ValidationException;

    Course removeCourse(String courseCode) throws ValidationException;

    void cleanCourseGrade(String courseCode) throws ValidationException;

    List<Course> getAll();

}
```

The case study

Conclusion

- Well-organized project
- Easier to locate the issue
- Easier to comprehend and start coding
- Easier to find and re-use a logic

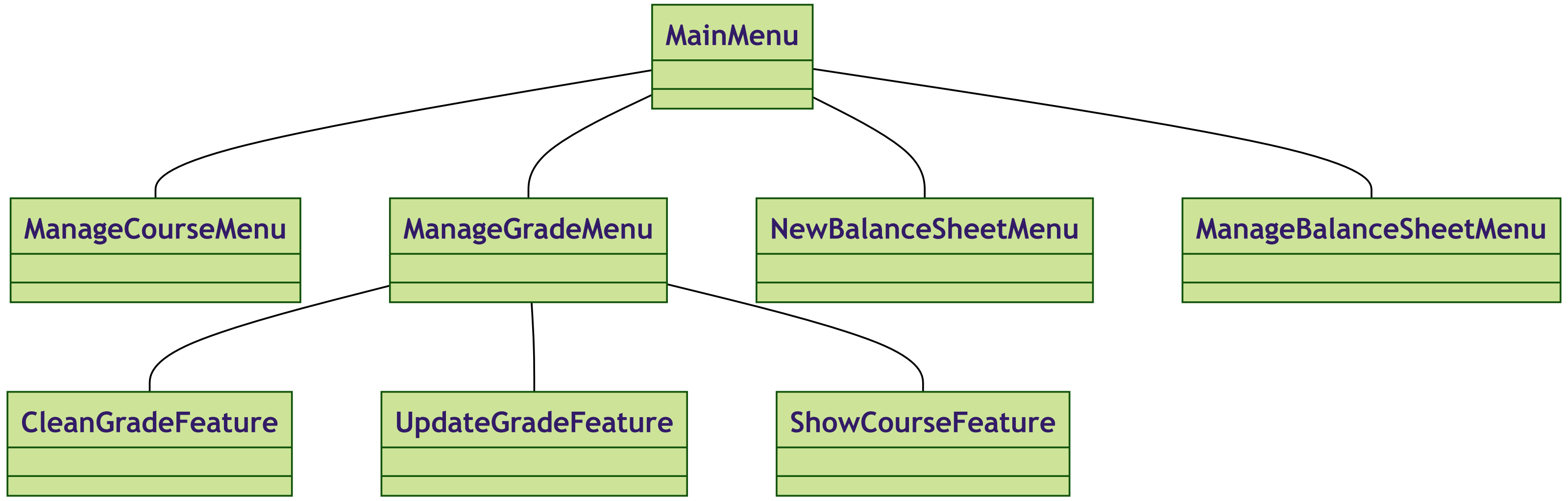


```java
package com.netcompany.utils;

import java.time.Instant;
import java.util.regex.Pattern;

public class StringUtils {
    static final Pattern decimalPattern = Pattern.compile("^\\d+(.\\d+)?$");

    public static String padRight(String s, int n) {
        return String.format("%-" + n + "s", s);
    }

    public static String padLeft(int value, int n) {
        return padLeft(String.valueOf(value), n);
    }

    public static String padLeft(String s, int n) {
        return String.format("%" + n + "s", s);
    }

    public static boolean isMatchedDecimalPattern(String decimalValue) {
        return decimalPattern.matcher(decimalValue).find();
    }

    public static String toEpocTimeString(long epochMilli) {
        return Instant.ofEpochMilli(epochMilli).toString();
    }
}
```

The case study

Conclusion

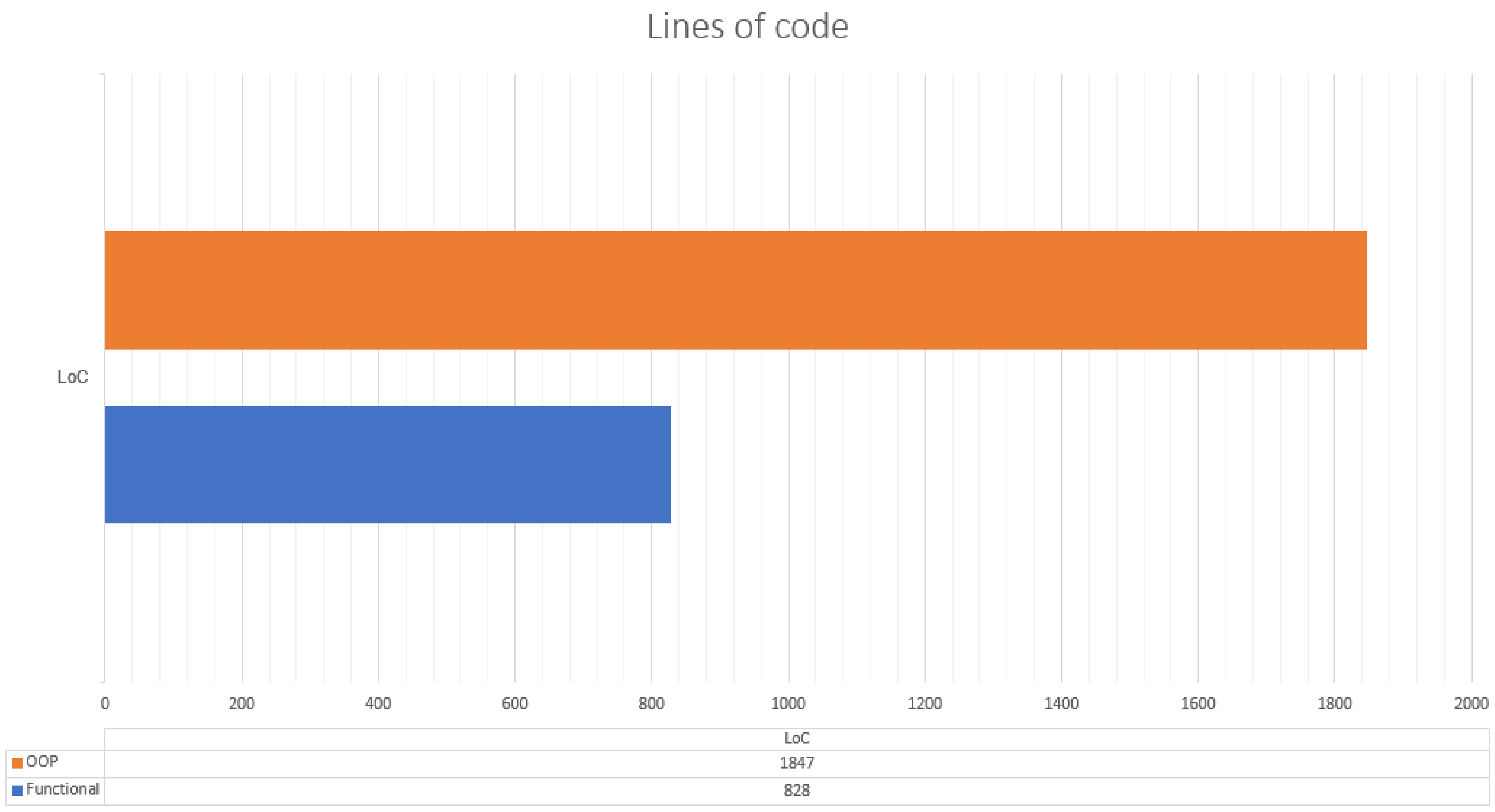
- Well-organized project
- Easier to locate the issue
- Easier to comprehend and start coding
- Easier to find and re-use a logic
- Loosing the coupling and increase cohesion



The case study

Conclusion

- Well-organized project
- Easier to locate the issue
- Easier to find and re-use a logic
- Loosing the coupling and increase cohesion
- Increase the complexity

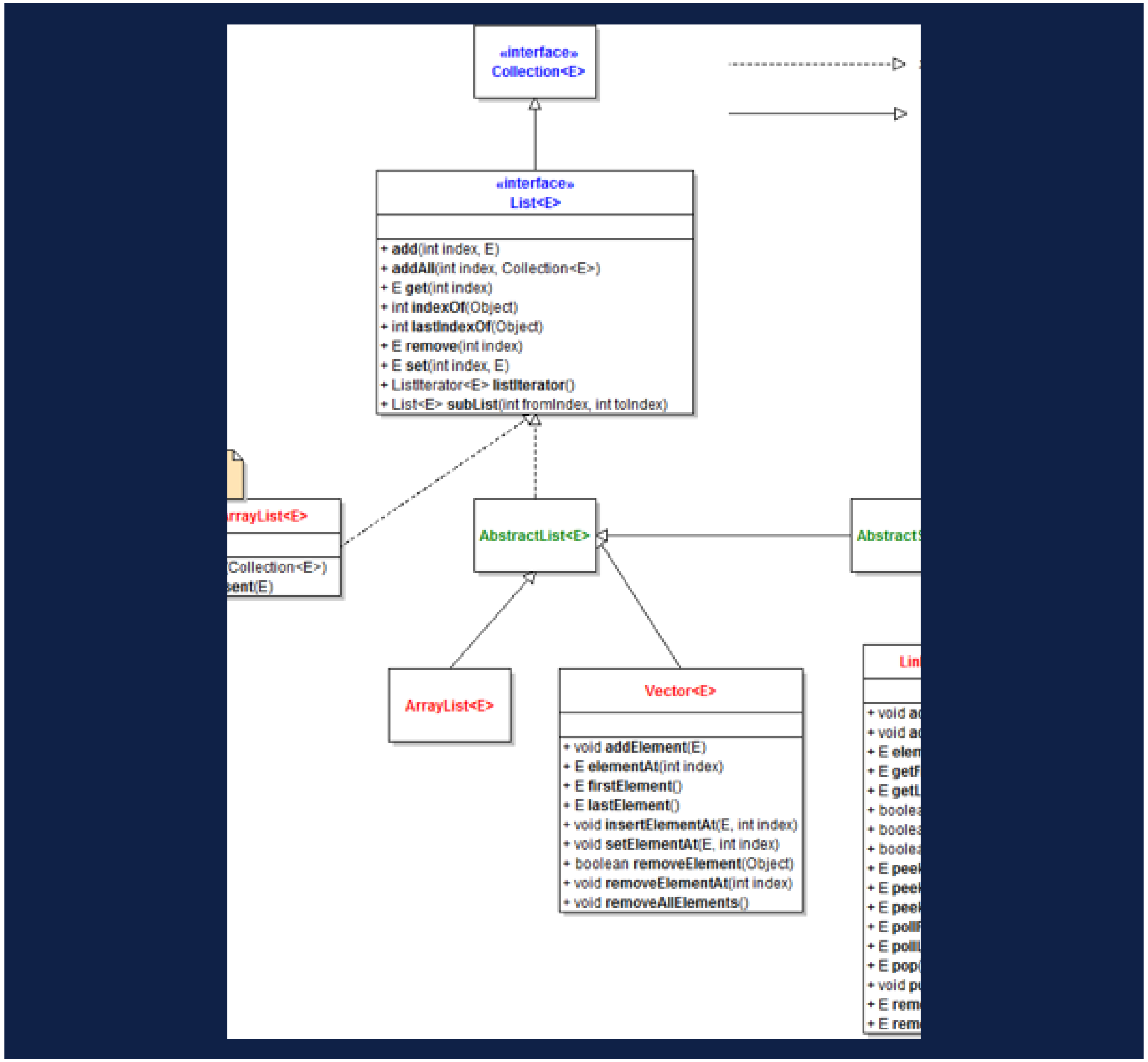


EXAMPLE FROM REAL PROJECTS

netcompany

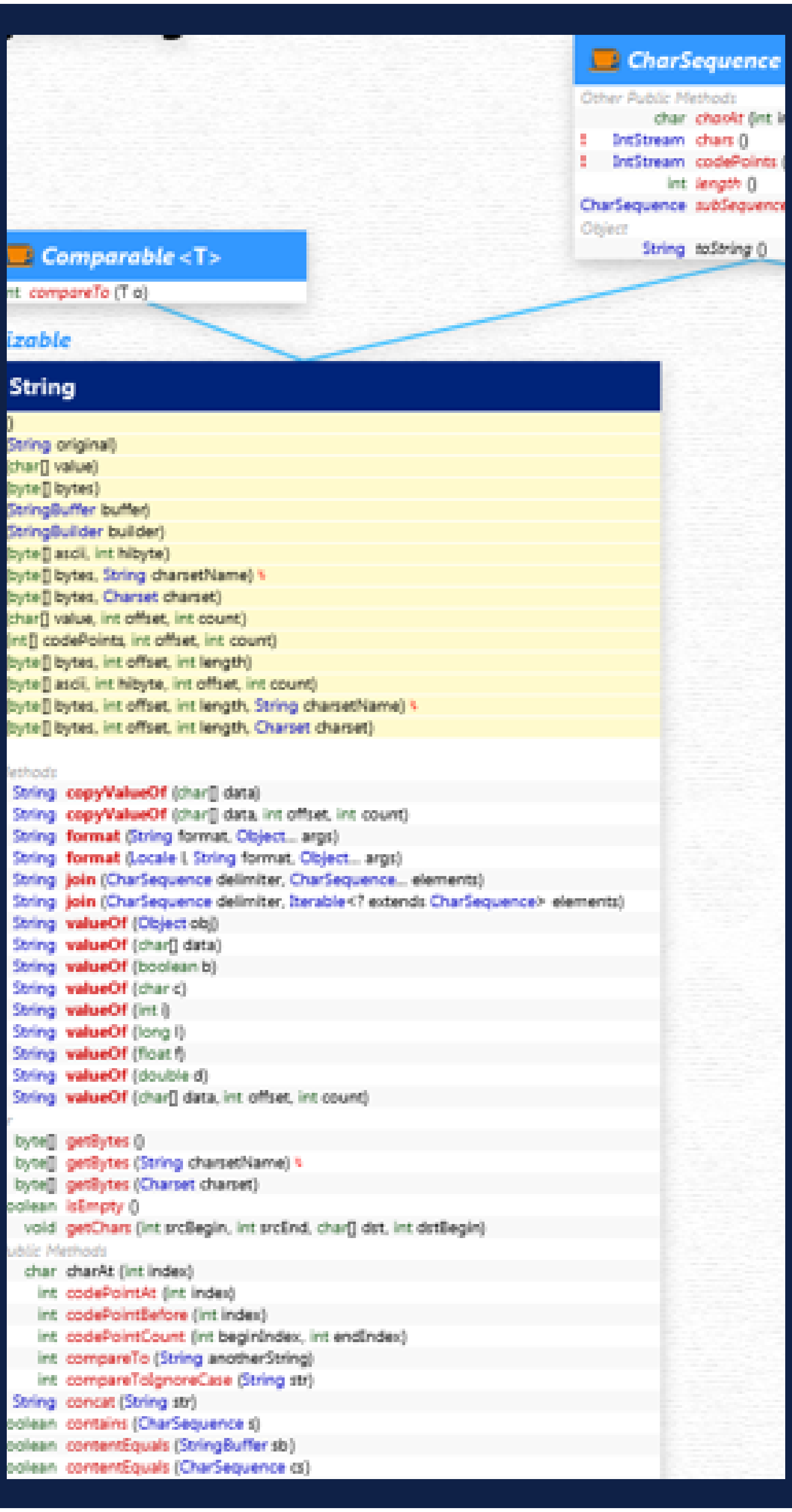
Example from real projects

Java List classes hierarchy



Example from real projects

Java String class



```
CharSequence
Other Public Methods
          char  charAt (int i
!    IntStream  chars ()
!    IntStream  codePoints (
           int  length ()
CharSequence  subSequence
Object
        String  toString ()
```

```
Comparable <T>
nt  compareTo (T o)
```

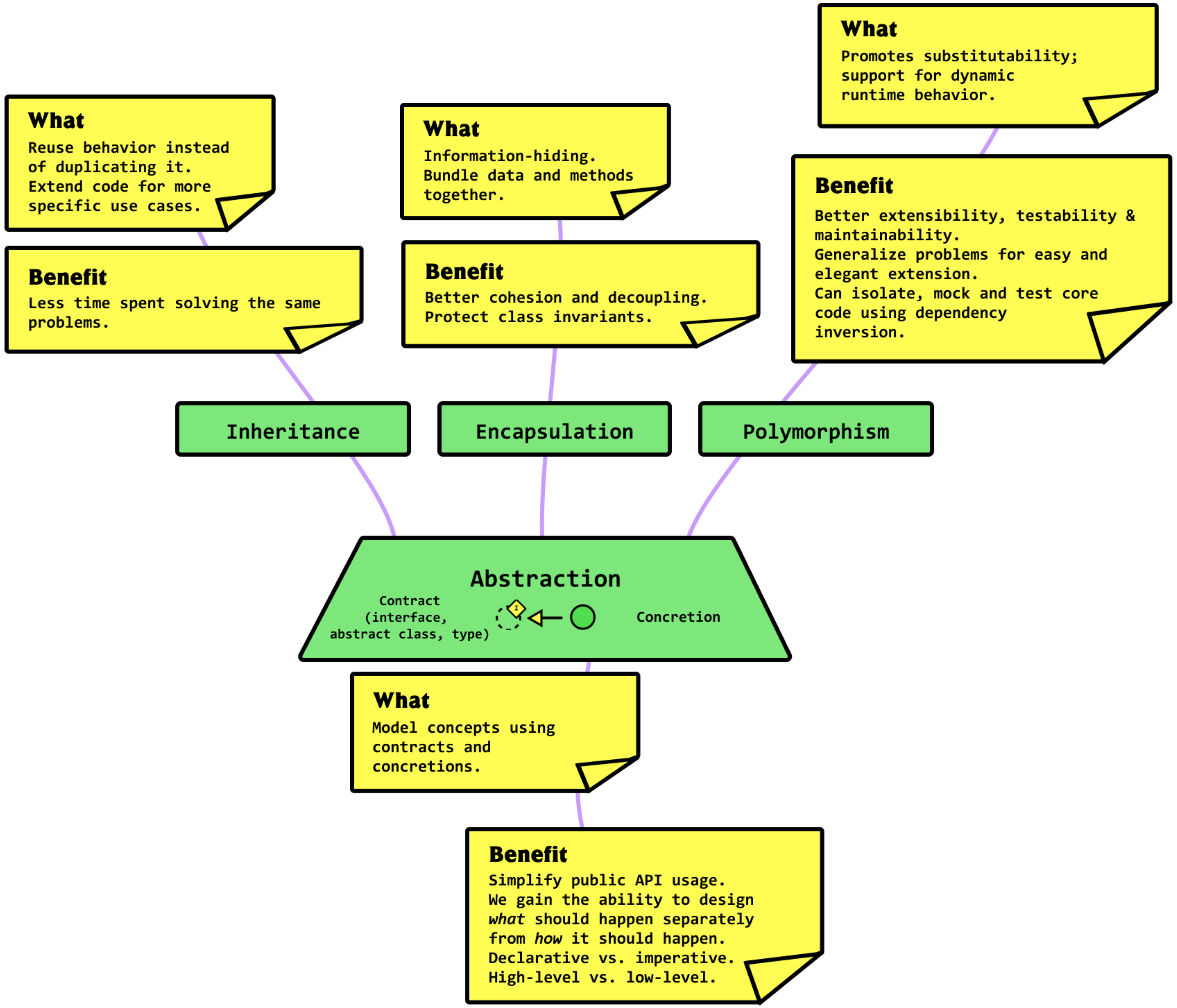
```
izable
```

```
String
)
String original)
char[] value)
byte[] bytes)
StringBuffer buffer)
StringBuilder builder)
byte[] ascii, int hibyte)
byte[] bytes, String charsetName)
byte[] bytes, Charset charset)
char[] value, int offset, int count)
int[] codePoints, int offset, int count)
byte[] bytes, int offset, int length)
byte[] ascii, int hibyte, int offset, int count)
byte[] bytes, int offset, int length, String charsetName)
byte[] bytes, int offset, int length, Charset charset)

ethods
 String  copyValueOf (char[] data)
 String  copyValueOf (char[] data, int offset, int count)
 String  format (String format, Object... args)
 String  format (Locale l, String format, Object... args)
 String  join (CharSequence delimiter, CharSequence... elements)
 String  join (CharSequence delimiter, Iterable<? extends CharSequence> elements)
 String  valueOf (Object obj)
 String  valueOf (char[] data)
 String  valueOf (boolean b)
 String  valueOf (char c)
 String  valueOf (int i)
 String  valueOf (long l)
 String  valueOf (float f)
 String  valueOf (double d)
 String  valueOf (char[] data, int offset, int count)
r
 byte[]  getBytes ()
 byte[]  getBytes (String charsetName)
 byte[]  getBytes (Charset charset)
oolean  isEmpty ()
   void  getChars (int srcBegin, int srcEnd, char[] dst, int dstBegin)
ublic Methods
   char  charAt (int index)
    int  codePointAt (int index)
    int  codePointBefore (int index)
    int  codePointCount (int beginIndex, int endIndex)
    int  compareTo (String anotherString)
    int  compareToIgnoreCase (String str)
 String  concat (String str)
oolean  contains (CharSequence s)
oolean  contentEquals (StringBuffer sb)
oolean  contentEquals (CharSequence cs)
```

SUMMARY

netcompany

Summary



What
Reuse behavior instead of duplicating it.
Extend code for more specific use cases.

Benefit
Less time spent solving the same problems.

What
Information-hiding.
Bundle data and methods together.

Benefit
Better cohesion and decoupling.
Protect class invariants.

What
Promotes substitutability; support for dynamic runtime behavior.

Benefit
Better extensibility, testability & maintainability.
Generalize problems for easy and elegant extension.
Can isolate, mock and test core code using dependency inversion.

What
Model concepts using contracts and concretions.

Benefit
Simplify public API usage.
We gain the ability to design *what* should happen separately from *how* it should happen.
Declarative vs. imperative.
High-level vs. low-level.

Summary

- The goal of software design is to write the code that:
  - meet the customer needs.
  - cost-effectively change by developer.
- The OOP is the guidance to design a robust application.
- The Abstraction is the foundation to build other OOP principles.
- The OOP is not just about design the domain objects.
- Practice OOP from fresher level will build up the system thinking.
- Apply OOP will make the project more complex and cost the runtime resources.

PRACTICAL

netcompany

Practical

Case study:

- Opal tower Parking Management System
- Netcompany Claim Application

Requirements

Use object-oriented programming to build a simple expense application.

- Identify actor(s) of this application.
- Identify objects, their properties and methods name.
- Describe relationship of objects, object hierarchy.
- Describe main functions.

Practical

Opal tower Parking Management System

Business: OPAL Tower is a modern building which has 41 floors and 4 basements, including high-class apartments, offices and convenient and modern shop-houses. The owner of building want to have a nice parking management application.

The building has 4 basements, the first 3 basements are preserved for residences and the last one are mixed for both residences, employees and visitors. Each basement has 100 parking slots for cars and a separated area for motorbikes. There is a LED panel at the entry displaying how many slots (for car) available at the moment.

For residences and employees, they have to register their vehicles beforehand by providing owner's information, vehicle info. These info will be stored in a small card used for check-in/check-out. User must present this card for check-in/out. For visitor, card will be provided when check-in and returned to parking attendant when check out Parking fee for residences and employees is a fix amount and charged monthly. For visitors, the fee is calculated by block rate and be payed at check-out by cash.

- For motorbike: 4k for first block 4 hours, 1k for each next hour.
- For car: 40k for first 4 hours, 20k for each next hour.

Practical

Netcompany Claim Application

Business: Netcompany supports multiple expenses for their employees such as: weekly breakfast, monthly transportation allowance, phone allowance … After spent money on these services, employees can claim back the money from Netcompany.

In order to help finance department manages all the claims, there is a need of an IT solution. The application has abilities below:

- Employees can create, input info (Draft) and submit (Submitted) the claim when they feel it's ok.
- There are some required info on each claim:
  - The amount of money
  - Date of claim
  - Image of the bill(s)
  - Expense types: Grab, Breakfast, Dinner, phone,…
  - Claim for whom(s)?

Practical

Netcompany Claim Application

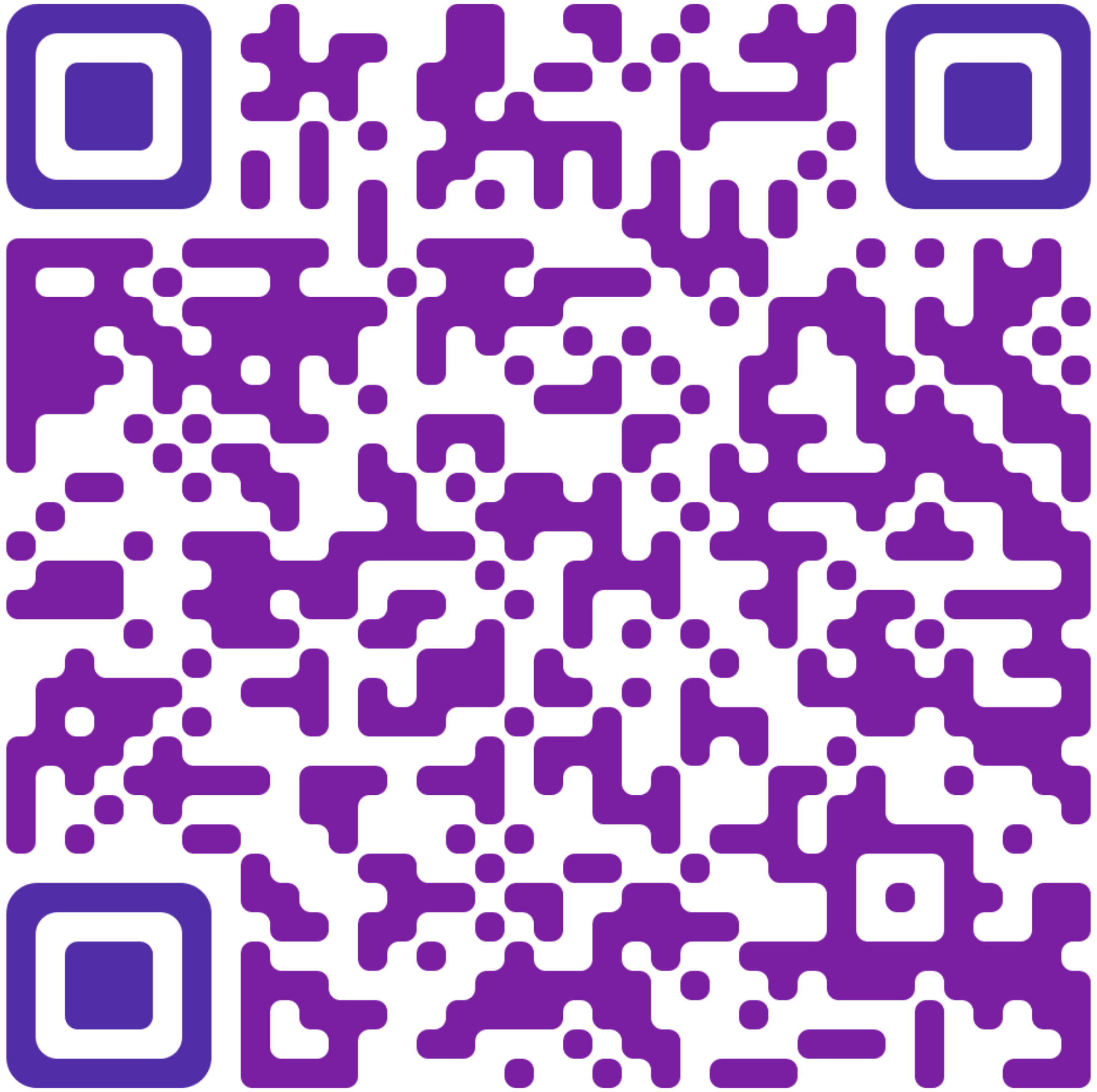
Business: Netcompany supports multiple expenses for their employees such as: weekly breakfast, monthly transportation allowance, phone allowance … After spent money on these services, employees can claim back the money from Netcompany.

In order to help finance department manages all the claims, there is a need of an IT solution. The application has abilities below:

- After claim submitted, the finance controller (associate, senior, manager) will review the claim(s), depends on the info of the claim, they can decide to approve (Approved) or reject (Rejected).
- If the amount of money is more than 10 million, only Senior Finance Controller has permission to approve it.
- If the amount of money is more than 100 million, only Manager Finance Controller has permission to approve it.

Practical

Opal tower Parking Management System



Netcompany Claim Application