Code Smells and Code Metrics

CS6.401 Software Engineering

Dr. Karthik Vaidhyanthan

karthik.vaidhyanathan@iiit.ac.in

https://karthikvaidhyanathan.com

INTERNATIONAL INSTITUTE OF
INFORMATION TECHNOLOGY
HYDERABAD

Acknowledgements

The materials used in this presentation have been gathered/adapted/generate from various sources as well as based on my own experiences and knowledge

-- Karthik Vaidhyanathan

Sources:

1. Refactoring, Improving the design of existing code, Martin Fowler et al., 2000
2. Various research articles that have been duly cited

How to get insights from existing software systems?

Mining Software Repositories

*The MSR field **analyzes rich data** available in **software repositories** to extract useful and **actionable information** about software projects and systems – msrconf.org*

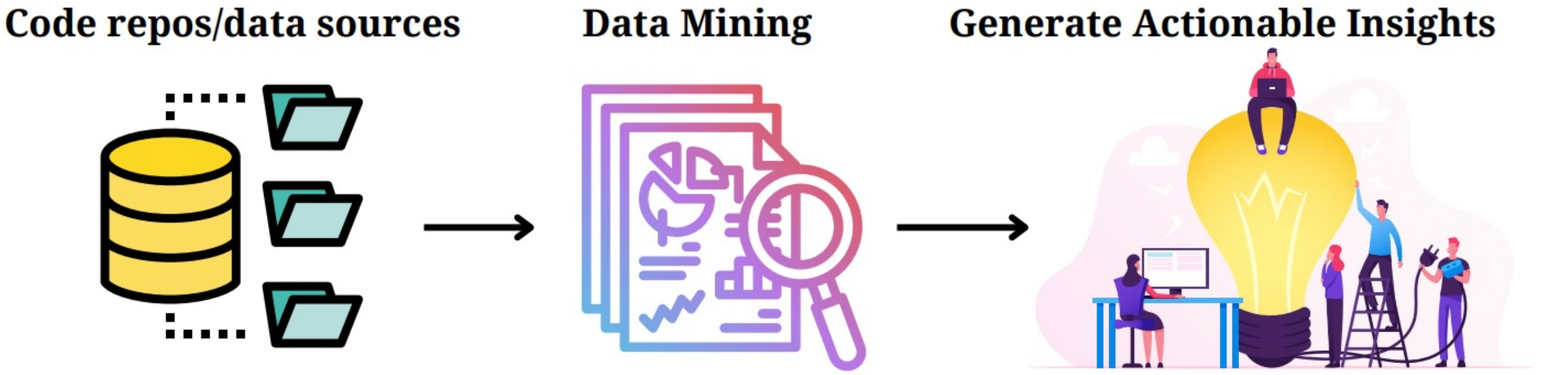
- Large amount of artefacts are generated in the software development process

- These data are available from various sources
  - Version control systems (SVN, Mercurial,..)
  - Q&A Forums (Stackoverflow, Stackexchange, etc.)
  - Bug repositories (BugZilla, Jira)
  - Code repositories (Github, Gitlab, etc.)
  - Crash reports, log files, execution traces, etc.

Mining Software Repositories

Repositories are great sources of unbiased data on how a product came to be - something that's very valuable and hard to find.

-- Andreas Zeller



Code repos/data sources

Data Mining

Generate Actionable Insights

- Generate insights on best practices (eg: Sources of technical debt)
- Develop approaches for automated code completions, bug localization,..
- …

https://conf.researchr.org/home/msr-2023

Source of quote: Walter Tichy. 2010. An Interview with Prof. Andreas Zeller: Mining your way to software reliability. Ubiquity 2010, November, Article 3 (November 2010), 6 pages.

But what about code?

Code can also Smell

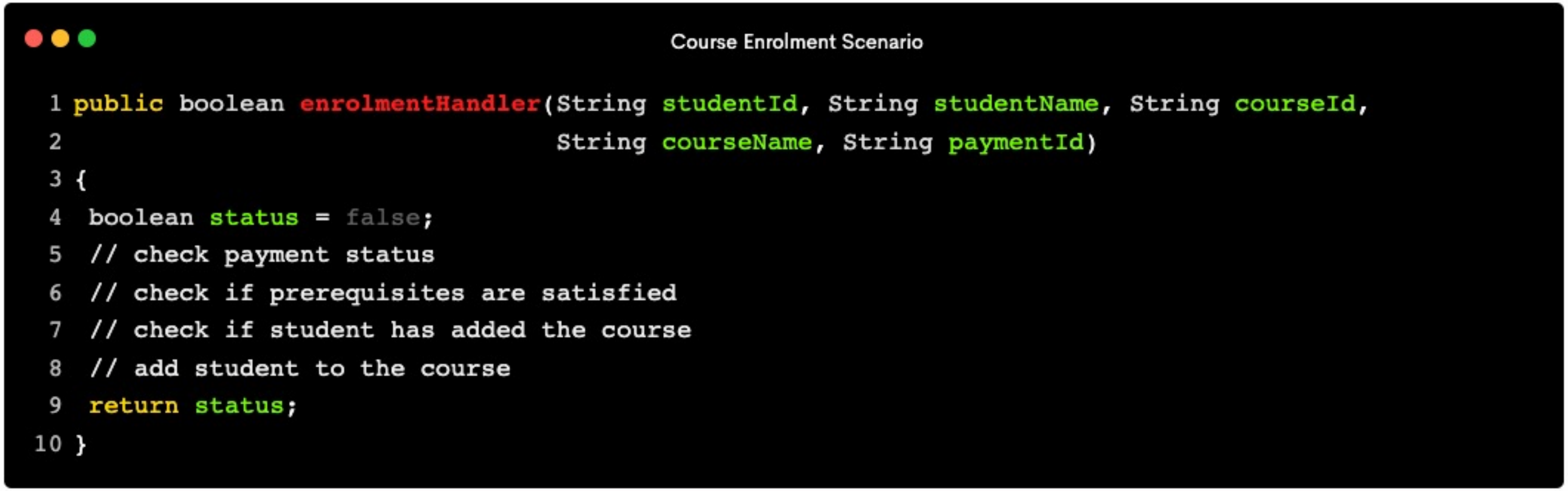
Code smell serves an indication that there is deeper problem in the system

- Code smells are only hints – not necessarily a problem!
- Look for bad smells – definitely needs a refactoring!
- But is there some list that could be used? Or common ones?

Image source: refactoring.guru

Some Examples

Code Smells – Long Method

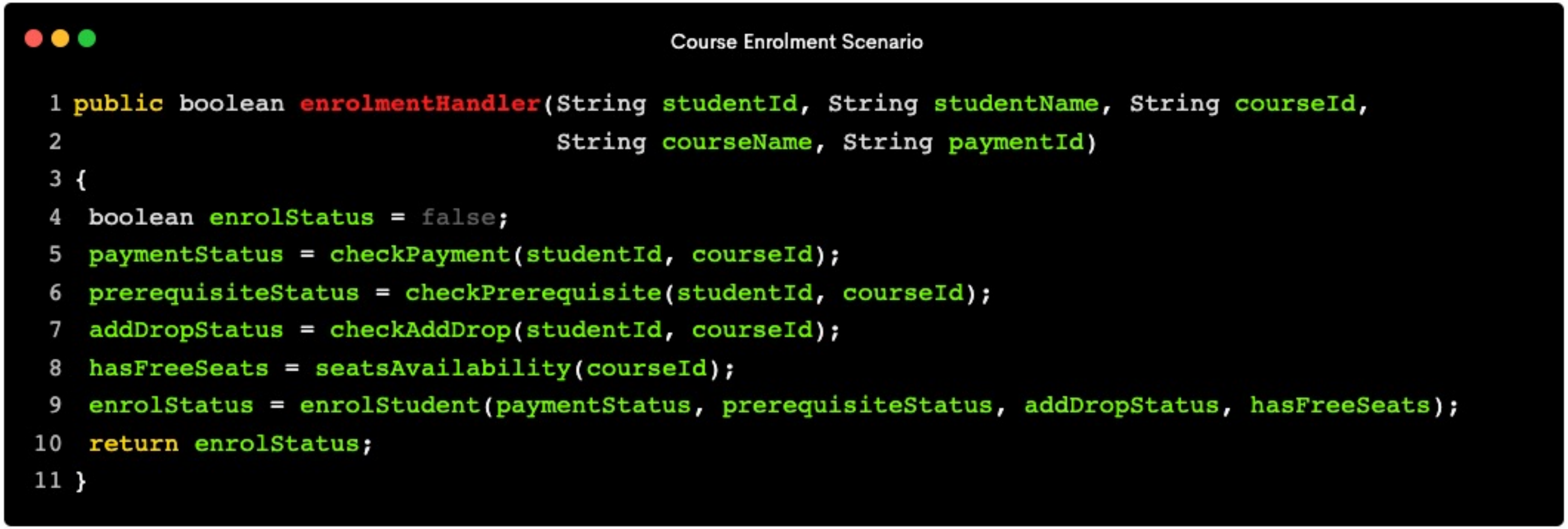


Course Enrolment Scenario

```
public boolean enrolmentHandler(String studentId, String studentName, String courseId,
                                String courseName, String paymentId)
{
 boolean status = false;
 // check payment status
 // check if prerequisites are satisfied
 // check if student has added the course
 // add student to the course
 return status;
}
```

Expand the comments to code – that’s a very long method!!

Long Method - Refactoring



```
public boolean enrolmentHandler(String studentId, String studentName, String courseId,
                                String courseName, String paymentId)
{
 boolean enrolStatus = false;
 paymentStatus = checkPayment(studentId, courseId);
 prerequisiteStatus = checkPrerequisite(studentId, courseId);
 addDropStatus = checkAddDrop(studentId, courseId);
 hasFreeSeats = seatsAvailability(courseId);
 enrolStatus = enrolStudent(paymentStatus, prerequisiteStatus, addDropStatus, hasFreeSeats);
 return enrolStatus;
}
```

What changes do you notice? – Any comments ? Can we do better?

Code Smells - Long Method

Longer a procedure, more difficult it is to understand

Context: A method has to perform a series of computations to accomplish a functionality

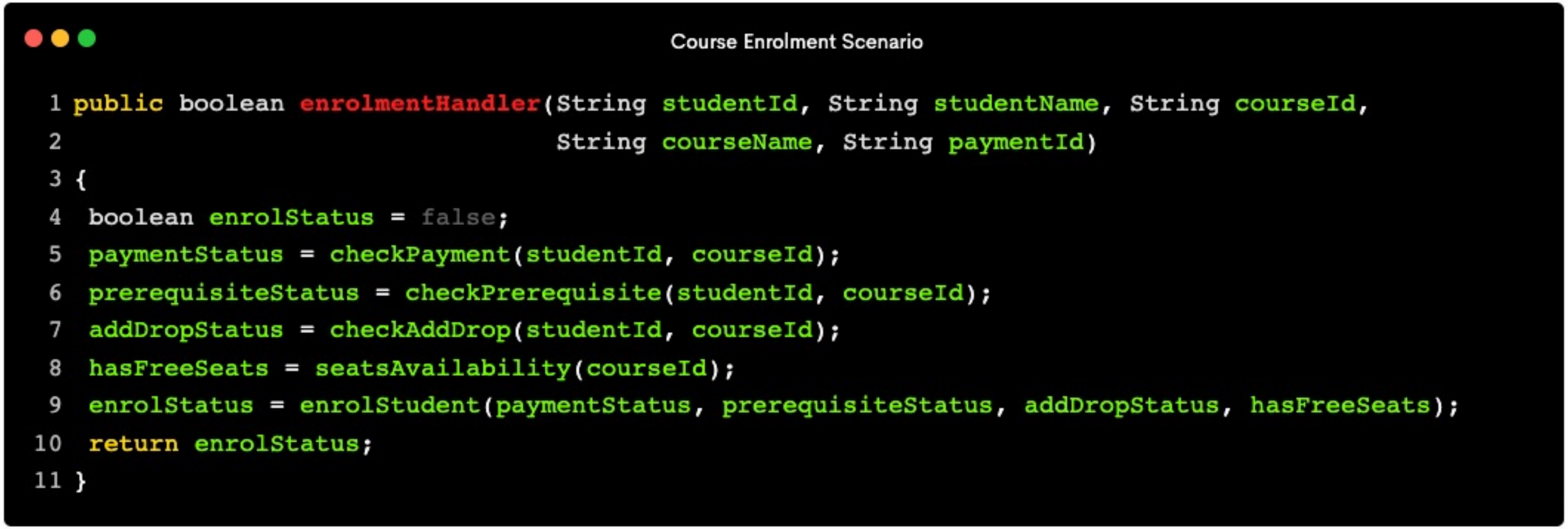
Problem: All the computations are written in its entirety inside one method making the method long - Too many lines of code!!

Suggested Refactoring: Any method longer than 20 lines (even 10+) is worth inspecting

Decompose the method into smaller methods -> Extract Method

This may result in other smells, which can call for further refactoring – parameter list

Code Smells – Long Parameter List



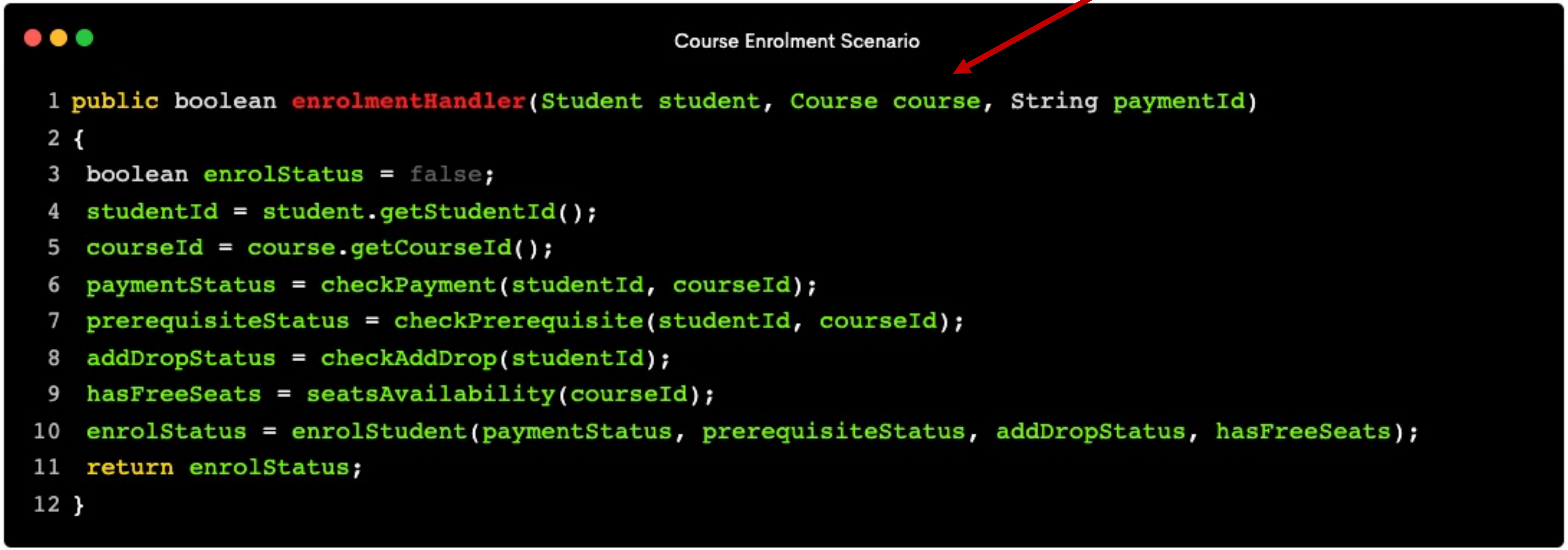
Course Enrolment Scenario

```
public boolean enrolmentHandler(String studentId, String studentName, String courseId,
                                String courseName, String paymentId)
{
 boolean enrolStatus = false;
 paymentStatus = checkPayment(studentId, courseId);
 prerequisiteStatus = checkPrerequisite(studentId, courseId);
 addDropStatus = checkAddDrop(studentId, courseId);
 hasFreeSeats = seatsAvailability(courseId);
 enrolStatus = enrolStudent(paymentStatus, prerequisiteStatus, addDropStatus, hasFreeSeats);
 return enrolStatus;
}
```

See the number of parameters that are taken as input by enrolmentHandler

Long Parameter List – Refactoring

Check parameters



```java
public boolean enrolmentHandler(Student student, Course course, String paymentId)
{
 boolean enrolStatus = false;
 studentId = student.getStudentId();
 courseId = course.getCourseId();
 paymentStatus = checkPayment(studentId, courseId);
 prerequisiteStatus = checkPrerequisite(studentId, courseId);
 addDropStatus = checkAddDrop(studentId);
 hasFreeSeats = seatsAvailability(courseId);
 enrolStatus = enrolStudent(paymentStatus, prerequisiteStatus, addDropStatus, hasFreeSeats);
 return enrolStatus;
}
```

The parameters are now respective objects, can we do further?

Code Smells - Long Parameter List

Try to have not more than 4 parameters – Not a Golden rule

Context: A method has to perform a series of computations to accomplish a functionality and it takes in lot of parameters

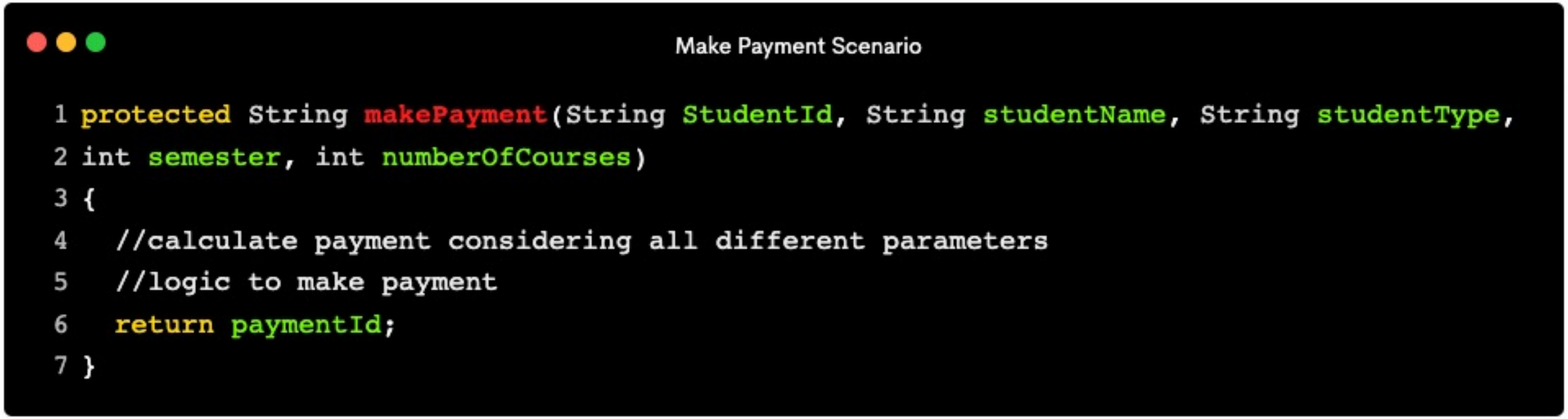
Problem: Hard to understand and the calling function/method needs to place the parameters in right positions, attracts adding of even more parameters!!

Suggested Refactoring: Multiple ways

- Replace parameter with method (call inside)
- Preserve whole object
- Introduce parameter object (if data items are related and no logical object exists

Long parameter can indicate other smells (eg: long methods, data clumps, primitive obsession)

Code Smells – Primitive Obsession



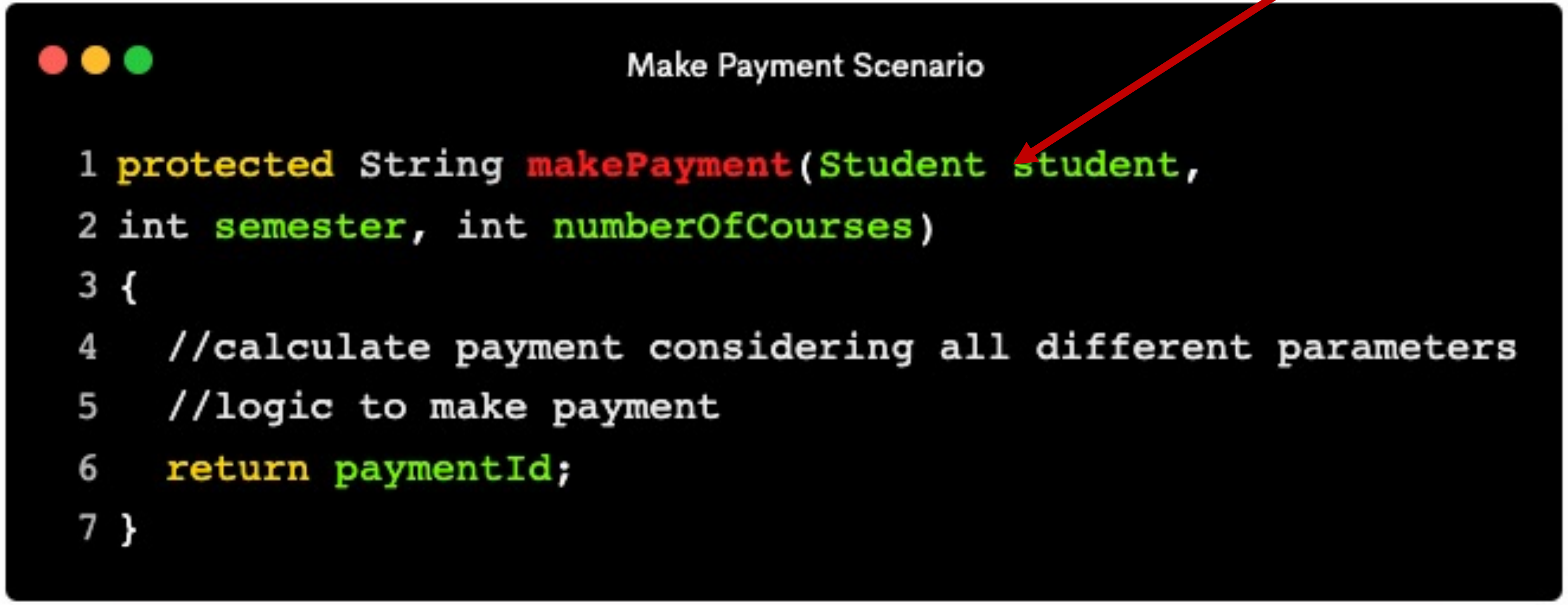
Make Payment Scenario

```
protected String makePayment(String StudentId, String studentName, String studentType,
int semester, int numberOfCourses)
{
  //calculate payment considering all different parameters
  //logic to make payment
  return paymentId;
}
```

Overuse of primitive types

Primitive Obsession – Refactoring

Check type of parameters



Make Payment Scenario

```
protected String makePayment(Student student,
int semester, int numberOfCourses)
{
  //calculate payment considering all different parameters
  //logic to make payment
  return paymentId;
}
```

We can do further, can we?

Code Smells – Primitive Obsession

Over use of primitive data types instead of objects

Context: A method has to perform a series of computations to accomplish a functionality and it takes in lot of parameters of primitive types

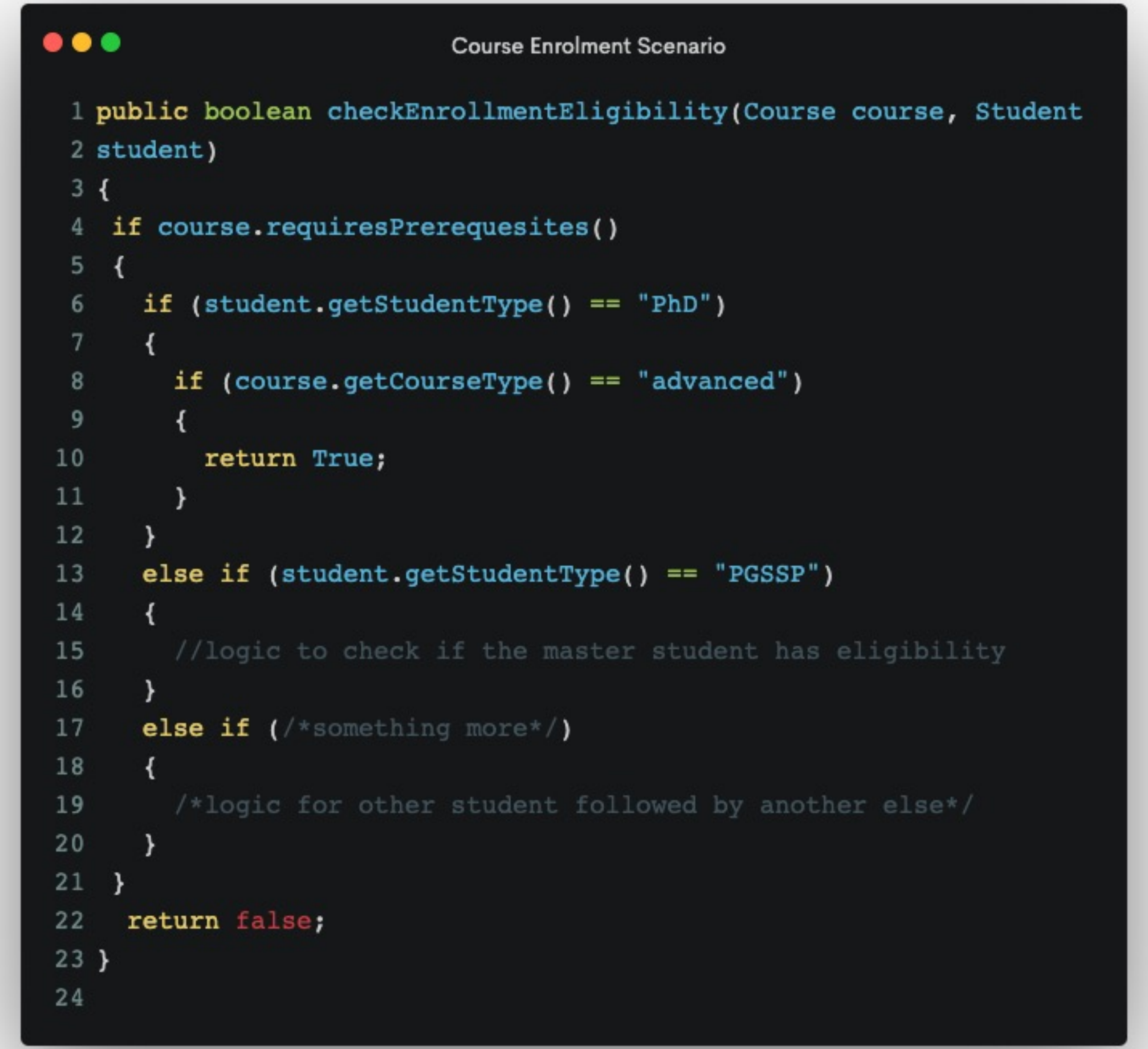
Problem: Having too many primitive types may lead to long parameters and can contribute to code duplication and type mismatches

Suggested Refactoring: Multiple ways

- Replace data value with object
- If there are group of fields (extract class)
- If there are fields that belong to object (Introduce parameter object)

Primitive obsession can lead to other smells (eg: long methods, data clumps, long Methods, etc.)

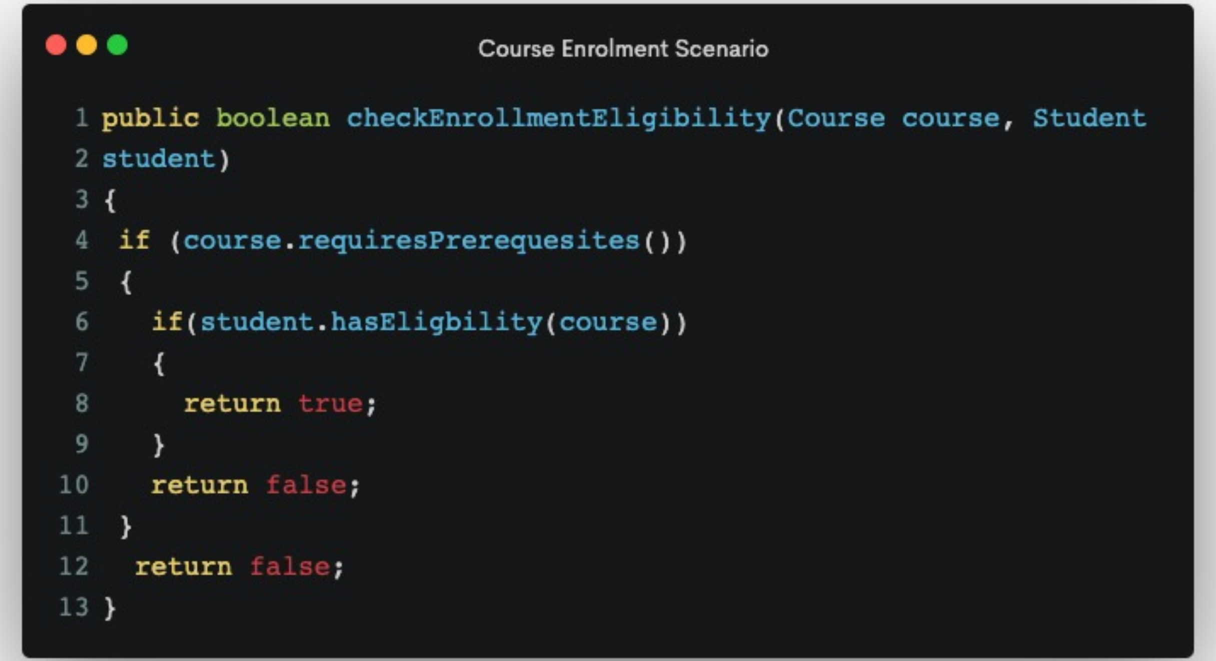
Code Smells – Switch Statements (Conditional Complexity)



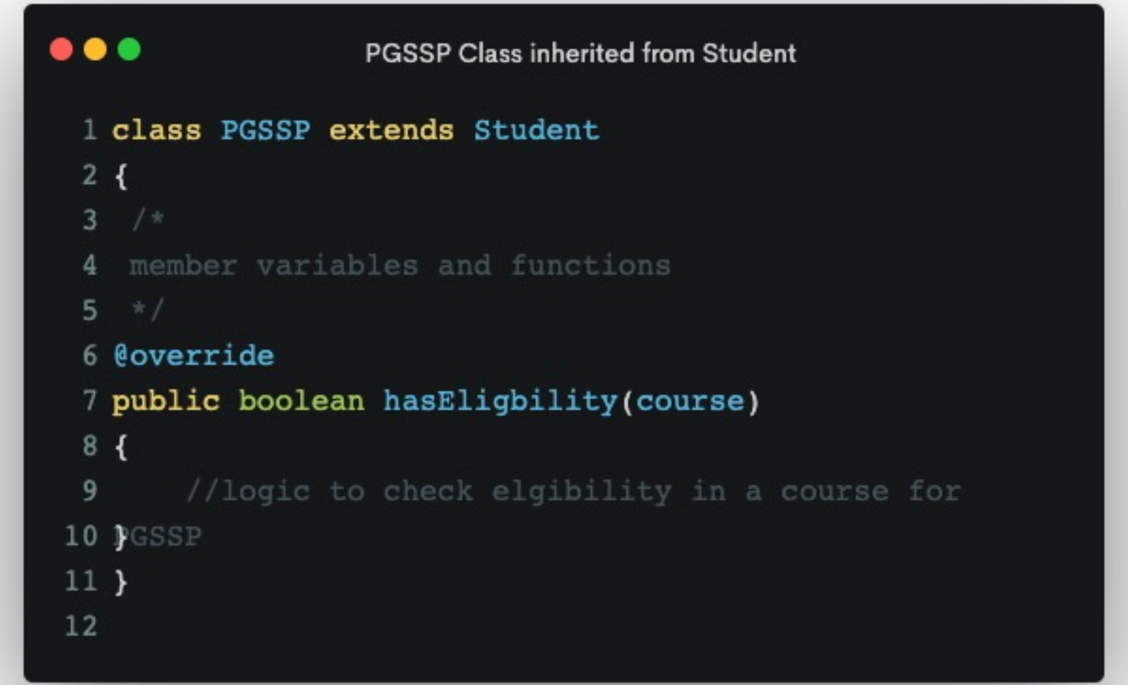
```
public boolean checkEnrollmentEligibility(Course course, Student
student)
{
 if course.requiresPrerequesites()
 {
   if (student.getStudentType() == "PhD")
   {
     if (course.getCourseType() == "advanced")
     {
       return True;
     }
   }
   else if (student.getStudentType() == "PGSSP")
   {
     //logic to check if the master student has eligibility
   }
   else if (/*something more*/)
   {
     /*logic for other student followed by another else*/
   }
 }
  return false;
}
```

Too many conditions!! Can we do better?

Switch Statements (Conditional Complexity) - Refactoring


Course Enrolment Scenario

```
public boolean checkEnrollmentEligibility(Course course, Student
student)
{
 if (course.requiresPrerequesites())
 {
   if(student.hasEligbility(course))
   {
     return true;
   }
   return false;
 }
  return false;
}
```


PGSSP Class inherited from Student

```
class PGSSP extends Student
{
 /*
 member variables and functions
 */
@override
public boolean hasEligbility(course)
{
    //logic to check elgibility in a course for
}PGSSP
}

```

Leveraging inheritance and polymorphism, we can do this for different types of student

Code Smells – Conditional Complexity

Complex set of switch or sequence of if conditions. When nesting goes too far!!

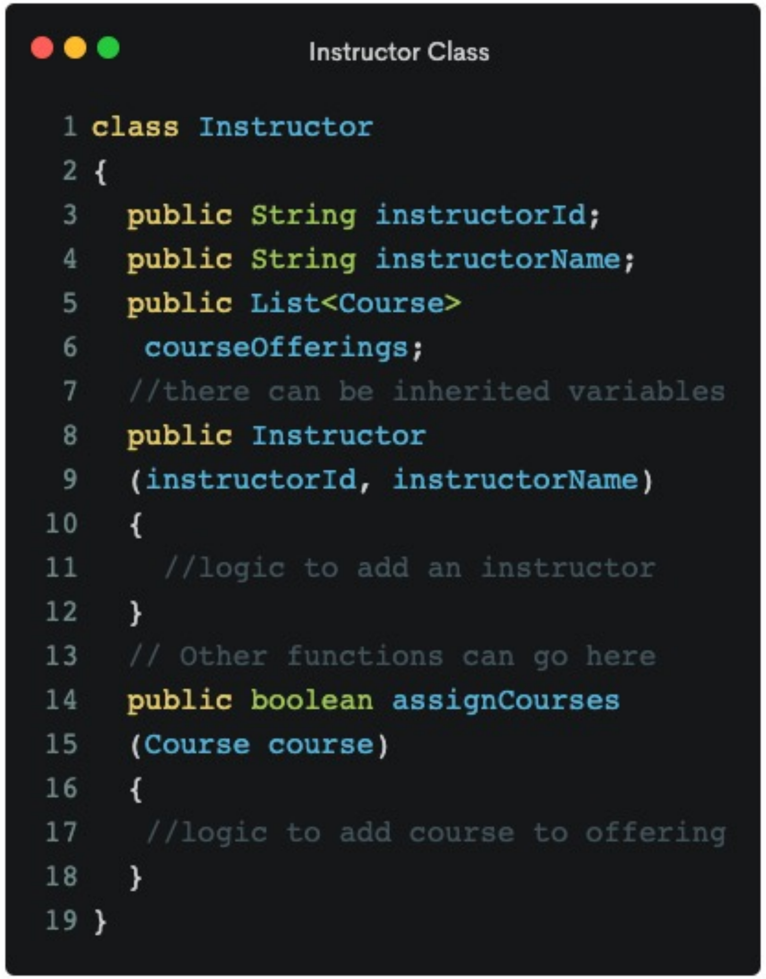
Context: A Single class has some operations and it requires editing multiple times when changes are made outside the class

Problem: Having too many conditional operations or switch makes it harder to understand, and high probability of breaking, testing also becomes difficult

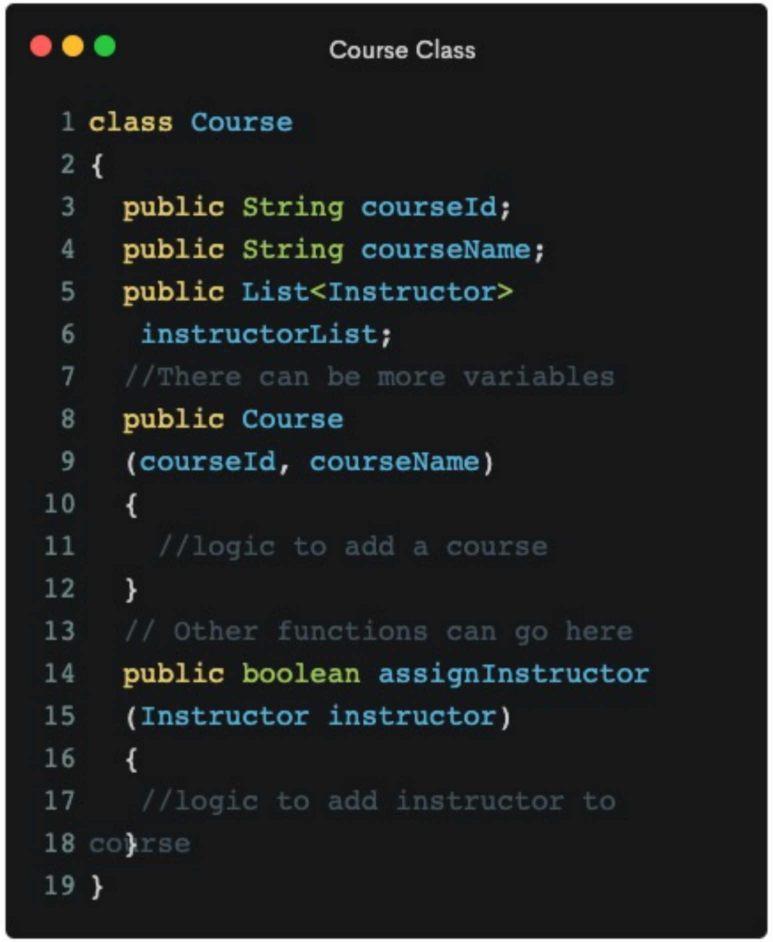
Suggested Refactoring: Multiple ways

- Introduce polymorphism
- Extract and move – Group things that need to be together, move to introduce polymorphism
- If there is only few cases that affect singe method – use extract method [Polymorphism can become overkill]

Code Smells – Divergent Change



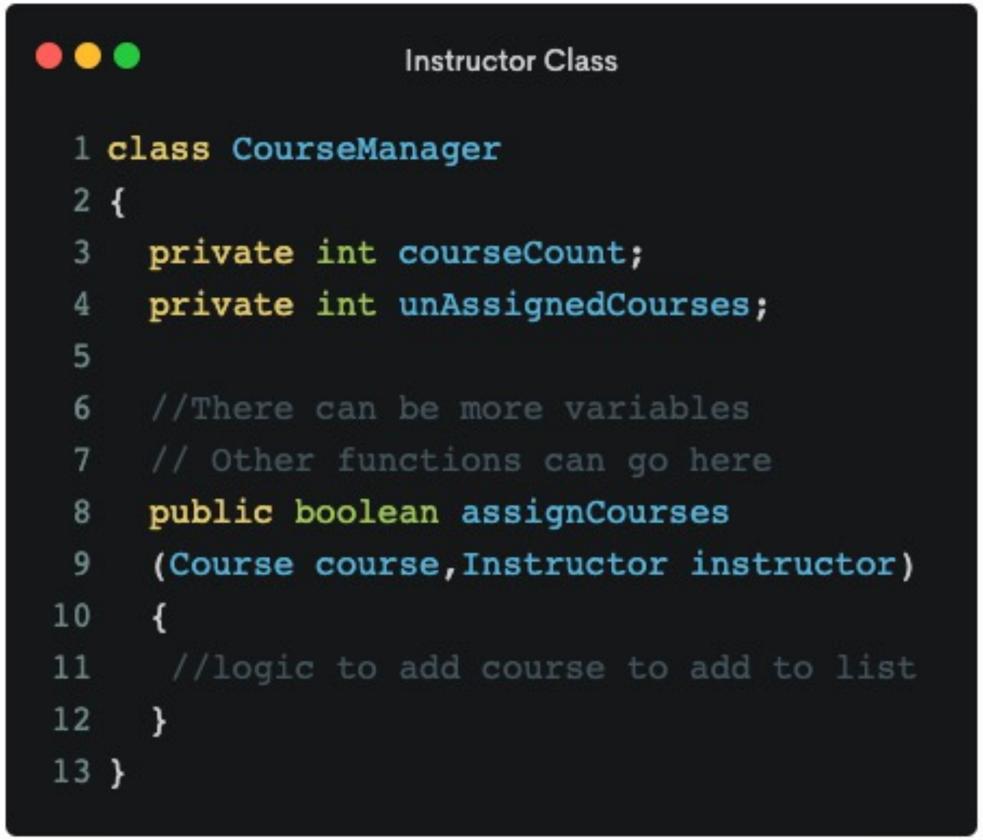
```
class Instructor
{
  public String instructorId;
  public String instructorName;
  public List<Course>
   courseOfferings;
  //there can be inherited variables
  public Instructor
  (instructorId, instructorName)
  {
    //logic to add an instructor
  }
  // Other functions can go here
  public boolean assignCourses
  (Course course)
  {
   //logic to add course to offering
  }
}
```



```
class Course
{
  public String courseId;
  public String courseName;
  public List<Instructor>
   instructorList;
  //There can be more variables
  public Course
  (courseId, courseName)
  {
    //logic to add a course
  }
  // Other functions can go here
  public boolean assignInstructor
  (Instructor instructor)
  {
   //logic to add instructor to
co}rse
}
```

What could be some issue here with respect to instructor class?

Divergent Change - Refactoring



```
class CourseManager
{
  private int courseCount;
  private int unAssignedCourses;

  //There can be more variables
  // Other functions can go here
  public boolean assignCourses
  (Course course,Instructor instructor)
  {
   //logic to add course to add to list
  }
}
```

Extract class and put the functionalities in one place so that one change does not impact others

Code Smells – Divergent Change

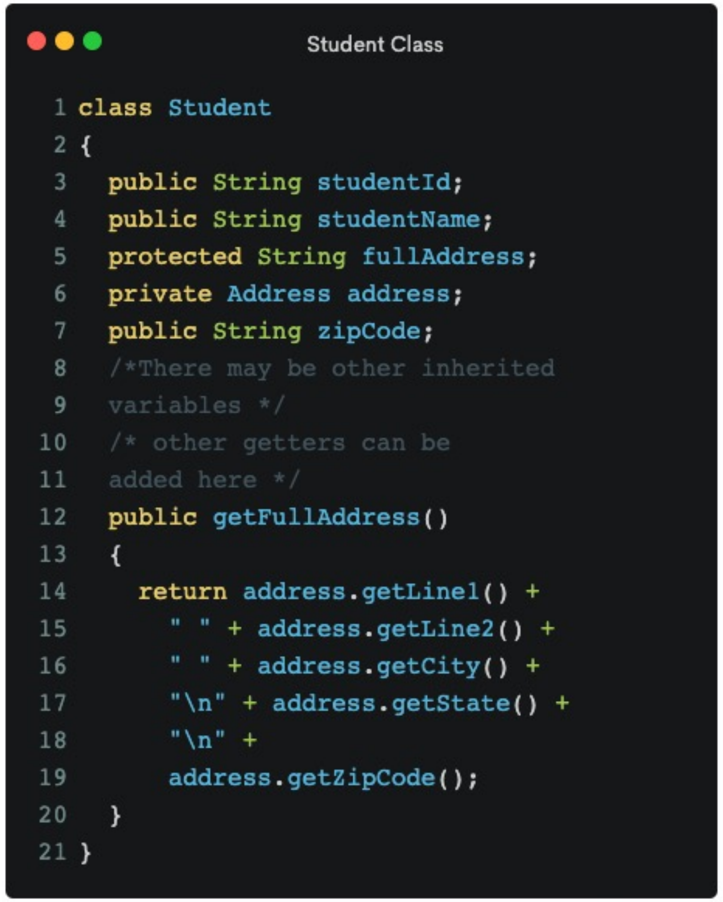
One change should not result in changes in "n" other places within a class

Context: A class has a method to perform an operation which is affected by changes happening in another method in same or different class

Problem: Impacts maintainability and results in a scenario where one needs to know everything. Also affects the testability and understandability

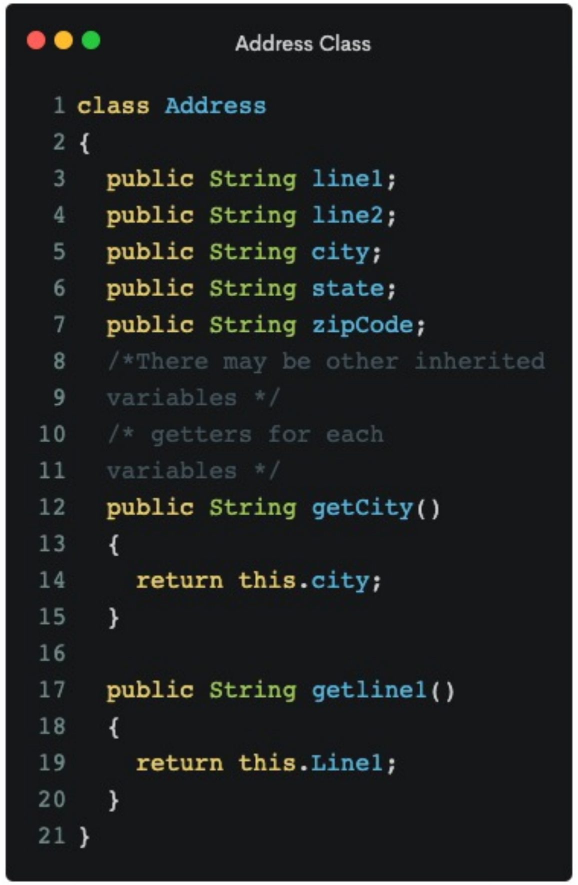
Suggested Refactoring: Multiple ways
- Extract Class – Put everything that changes together into one class
- Extract method – Check if the operations that change can be wrapped into a single method

Code Smells – Feature Envy



Student Class

```
class Student
{
  public String studentId;
  public String studentName;
  protected String fullAddress;
  private Address address;
  public String zipCode;
  /*There may be other inherited
  variables */
  /* other getters can be
  added here */
  public getFullAddress()
  {
    return address.getLine1() +
      " " + address.getLine2() +
      " " + address.getCity() +
      "\n" + address.getState() +
      "\n" +
      address.getZipCode();
  }
}
```



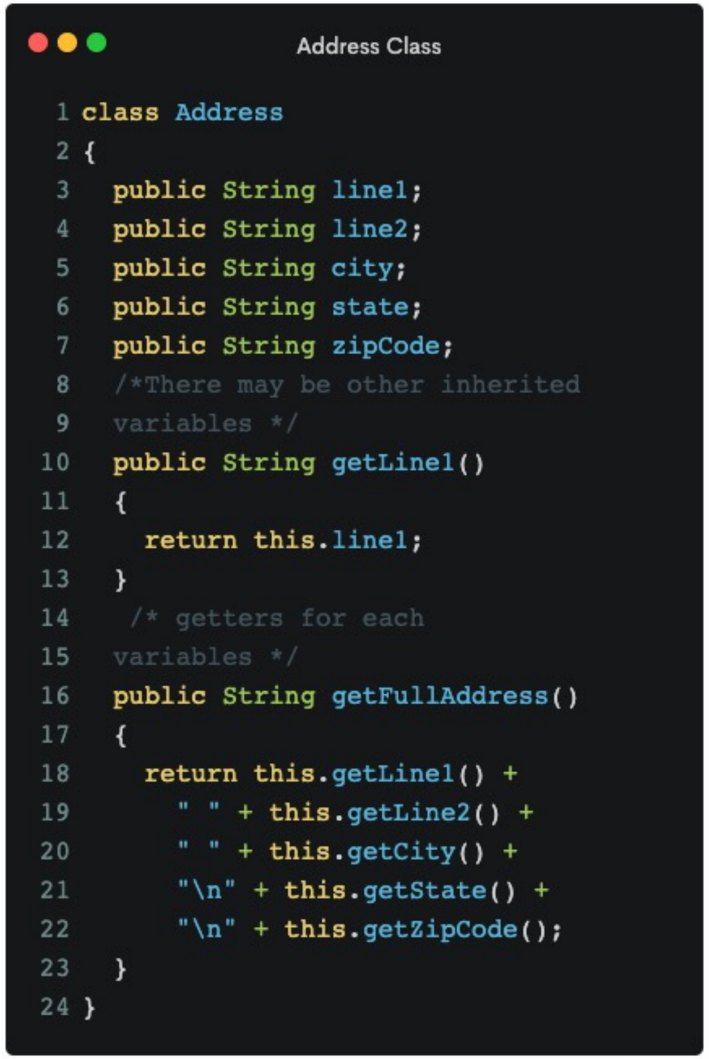
Address Class

```
class Address
{
  public String line1;
  public String line2;
  public String city;
  public String state;
  public String zipCode;
  /*There may be other inherited
  variables */
  /* getters for each
  variables */
  public String getCity()
  {
    return this.city;
  }

  public String getline1()
  {
    return this.Line1;
  }
}
```

One class depending too much on functions from another class – Envious!!

Feature Envy - Refactoring



Address Class

```
class Address
{
  public String line1;
  public String line2;
  public String city;
  public String state;
  public String zipCode;
  /*There may be other inherited
  variables */
  public String getLine1()
  {
    return this.line1;
  }
   /* getters for each
  variables */
  public String getFullAddress()
  {
    return this.getLine1() +
      " " + this.getLine2() +
      " " + this.getCity() +
      "\n" + this.getState() +
      "\n" + this.getZipCode();
  }
}
```

Move the method to the class so that the single responsibility principle is also ensured

Code Smells – Feature Envy

Method in a class is envious of features offered by other classes

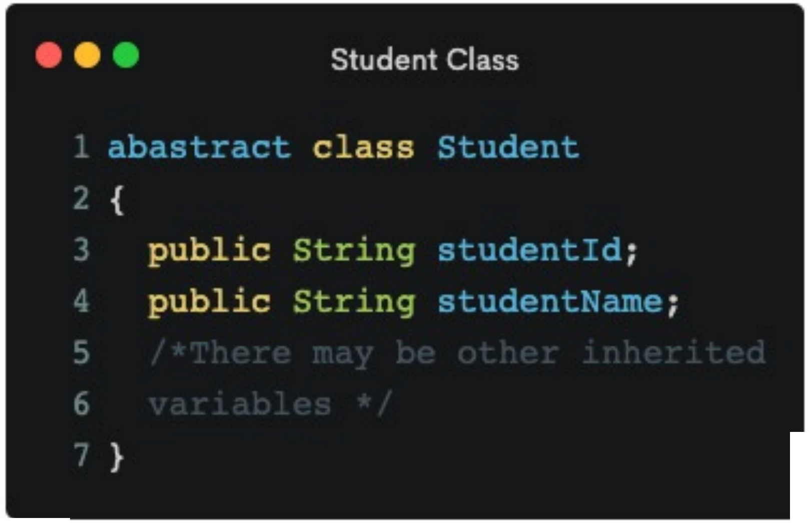
Context: A class has a method that needs to perform operations for which it depends on multiple data and operations in another class(es)

Problem: Causes coupling between different classes affecting extensibility and changeability. Testing also becomes challenging

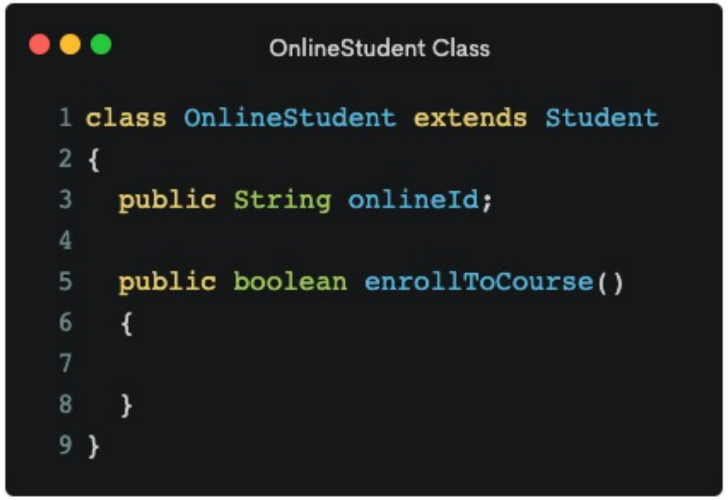
Suggested Refactoring: Multiple ways

- Move method – moving the method to where it belongs
- Extract and move method – When only part of the method has too much dependency

Code Smells – Speculative Generality



```
abastract class Student
{
  public String studentId;
  public String studentName;
  /*There may be other inherited
  variables */
}
```



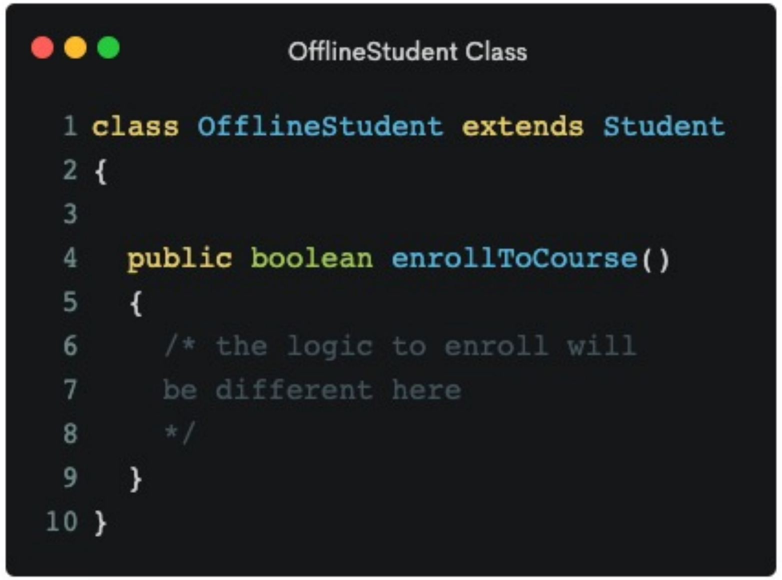
```
class OnlineStudent extends Student
{
  public String onlineId;

  public boolean enrollToCourse()
  {

  }
}
```

Sometimes we over design and overcomplicate things and speculate !!

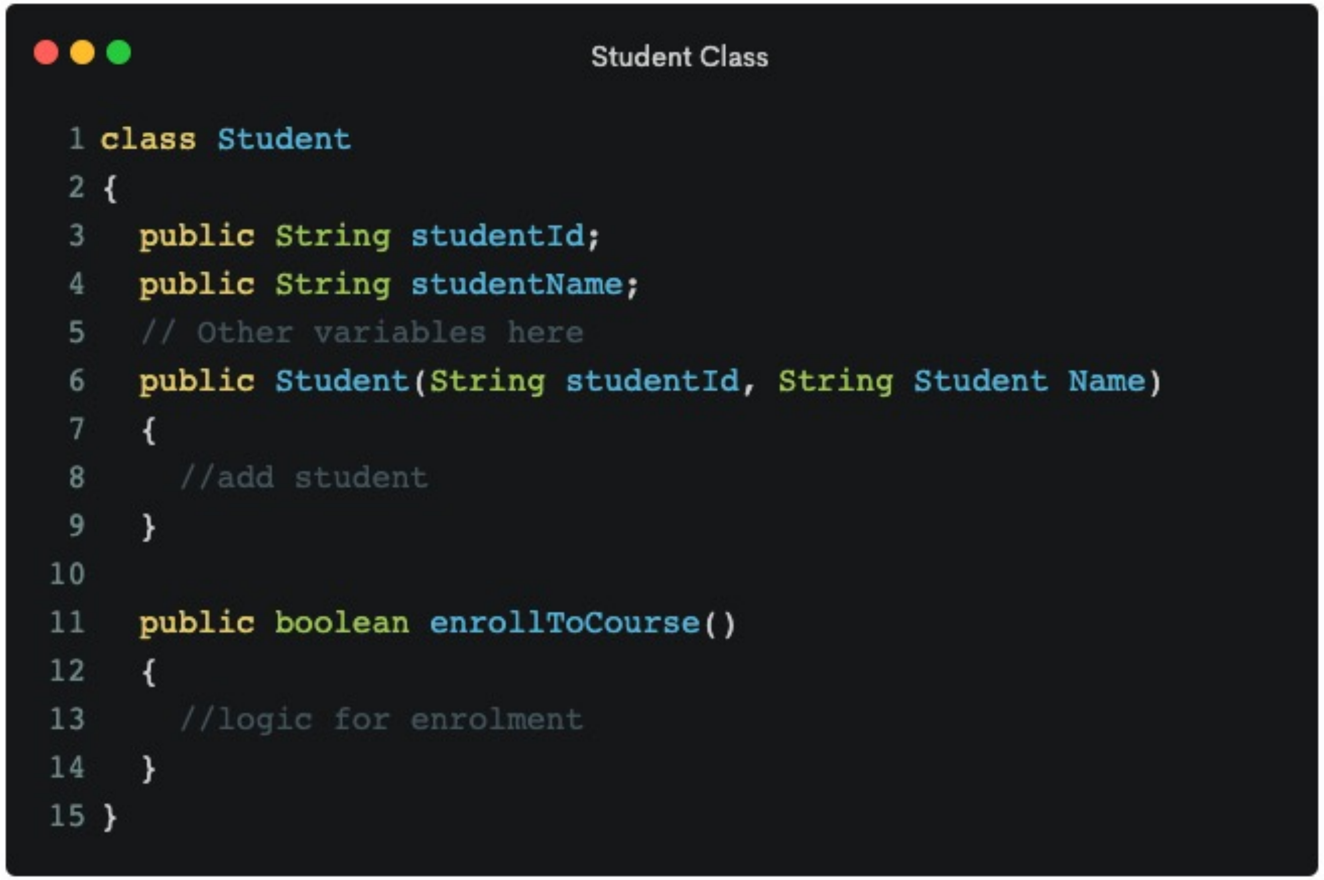
Do we need an inheritance at the point



```
class OfflineStudent extends Student
{

  public boolean enrollToCourse()
  {
    /* the logic to enroll will
    be different here
    */
  }
}
```

Speculative Generality - Refactoring



```
class Student
{
  public String studentId;
  public String studentName;
  // Other variables here
  public Student(String studentId, String Student Name)
  {
    //add student
  }

  public boolean enrollToCourse()
  {
    //logic for enrolment
  }
}
```

Refactoring by Collapsing the hierarchy

Code Smells – Speculative Generality

Code created with speculation that something will be required in feature and never implemented

Context: Classes have been built after extending classes creating inheritance relationship but never used or features have been planned but not implemented

Problem: Unwanted complexity affecting understandability. Can lead to some implementations in the child classes resulting in unwanted behaviour. Can be spotted when the only use of a class is some test cases.

Suggested Refactoring: Multiple ways

- Collapse Hierarchy – Remove abstract classes not doing much
- Use Inline Class – Remove unnecessary delegation
- Remove unused parameters
- Rename methods to be in line with context

Five Main Categories Of Smells

- **Bloaters** – Too many things packed, keeps accumulating (eg: Long Method, primitive obsession)

- **Object Oriented Abusers** – Incorrect use of OO principle or even incomplete (eg: Conditional complexity)

- **Change Preventers** – Changing one causes change in others (eg: Divergent Change)

- **Dispensables** – Code whose absence won't make a difference, but presence could! (eg: Speculative Generality)

- **Couplers** – Creates too much coupling between classes (eg: Feature Envy)

Quick Reference Cards

Smell Name	Short Description	Suggested Refactoring
Duplicated Code	Same code in more than one place	Extract Method, pull up field,..
Long Method	Too many things in one method	Extract Method, Decompose conditionals, Parameter objects,..
Large Class	One class is doing too much	Extract class, extract sub classes. Extract interface,..
Long Parameter List	Never ending list of parameters	Parameter object, Extract Method,..
Divergent Change	Too many changes in one class for different reasons	Extract class,..
Shortgun Surgery	One change => too many changes	Move Method, Move field, Inline class,..
Feature Envy	Interested in methods of another class	Extract Method, Move Method
Data Clumps	Same data items together in many places	Extract Class, Parameter Object, ..
Primitive Obsession	Using too many primitive data types	Extract class, Introduce parameter Object,..
Switch Statements	Complex switch statements, conditionals,..	Replace with explicit method, Replace conditional with polymorphism,..
Parallel Inheritance hierarchies	Requiring parallel subclasses creation	Move Method and Move field,..

Smell Name	Short Description	Refactoring
Lazy Class	Not doing much, exists there!	Collapse hierarchy,..
Speculative Generality	No use in the current context than test	Inline class, Collapse hierarchy,..
Temporary Field	Having instance variables not used much	Extract class for unused variables,..
Message Chains	Ask to one object which leads to next…	Hide Delegate, extract method and move method,..
Middle Man	Lots of delegations happening	Remove Middle Man, Inline Method, Replace delegate with inheritance,..
Inappropriate Hierarchy	Too much private information shared between classes	Extract class, hide delegate, Extract class, ..
Alternative Classes different Interface	Two classes having similar methods Using different interface	Move method, extract superclass,..
Incomplete Library Class	Modifying library class can be impossible	Introduce local extension, foreign method,..
Data Class	Classes with just some data fields	Move method, Extract method,..
Refused Bequest	Subclasses don’t need everything	Push down method, Push down field,..
Comments	Too much of comments is also bad	Rename method, Extract method,..

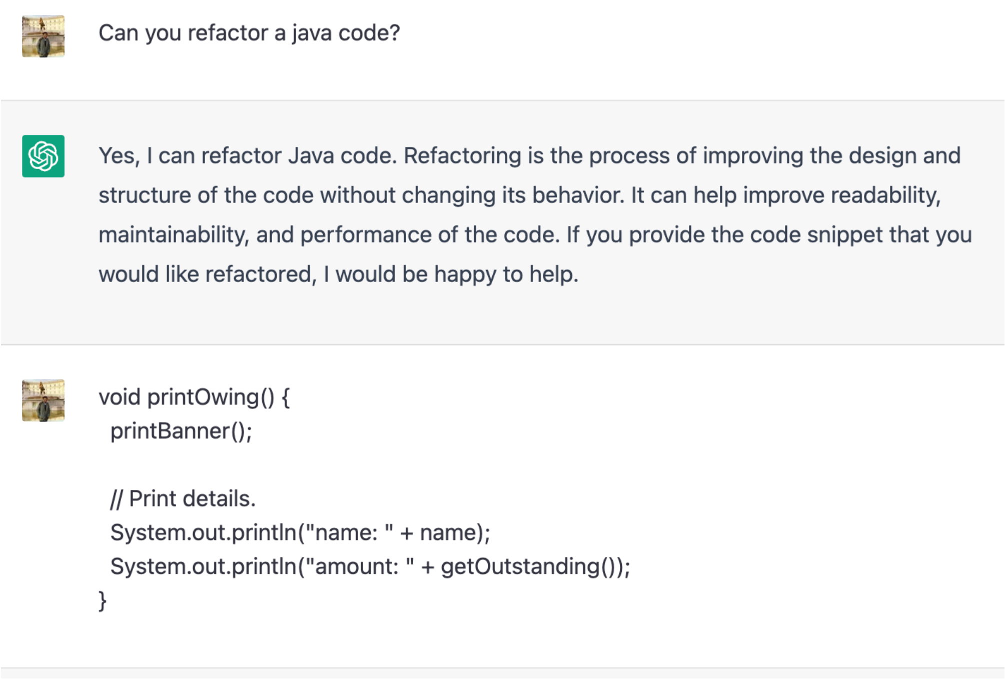
Refactoring - Best Practices

- Understand code well before refactoring
  - What you think might be a problem may not be a problem!
- Create tests and ensure the tests work just like before or even better
- Keep refactoring small and commit often – Take small steps, test and repeat
- The scope of refactoring needs to be defined clearly
  - Sometimes it can end in a loop
  - To do notes can be always useful
- It always helps when there are more eyes [Also for project!!]

Refactoring – Some Tools

- There is no single best tool that are available – Use your intuition along with existing tools
- IDEs provide a lot of support – Refactor menu (In IntelliJ IDEA)
- SonarLint – Like spellchecker for code, detects smells, checks for any possible issues (available for IDE's like IntelliJ, Eclipse, etc.)
- SonarQube – It's a server, given a repo it finds the list of code smells

ChatGPT-3 for refactoring – Just a try!

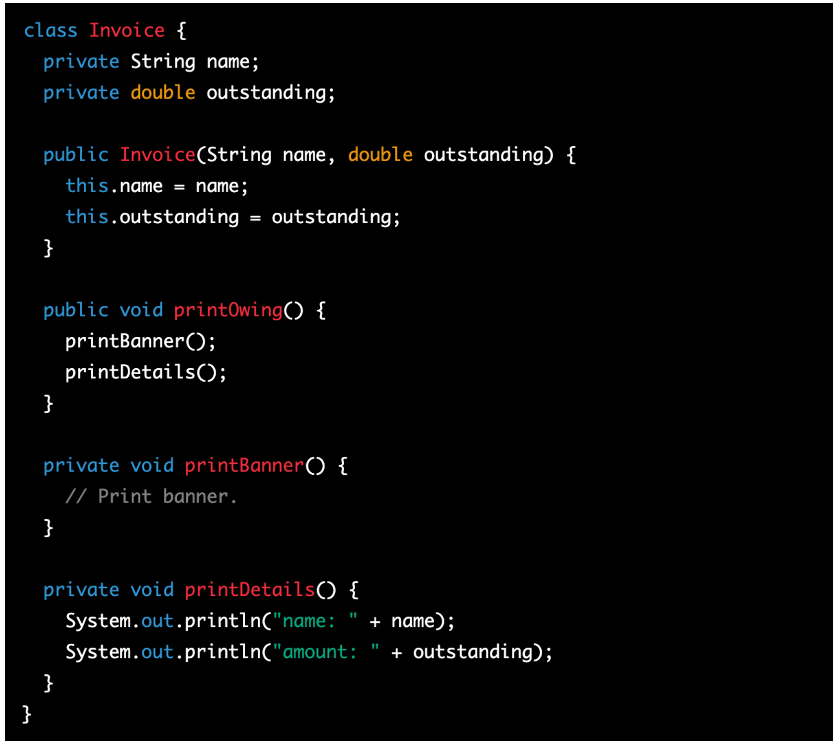


Can you refactor a java code?

Yes, I can refactor Java code. Refactoring is the process of improving the design and structure of the code without changing its behavior. It can help improve readability, maintainability, and performance of the code. If you provide the code snippet that you would like refactored, I would be happy to help.

```
void printOwing() {
  printBanner();

  // Print details.
  System.out.println("name: " + name);
  System.out.println("amount: " + getOutstanding());
}
```



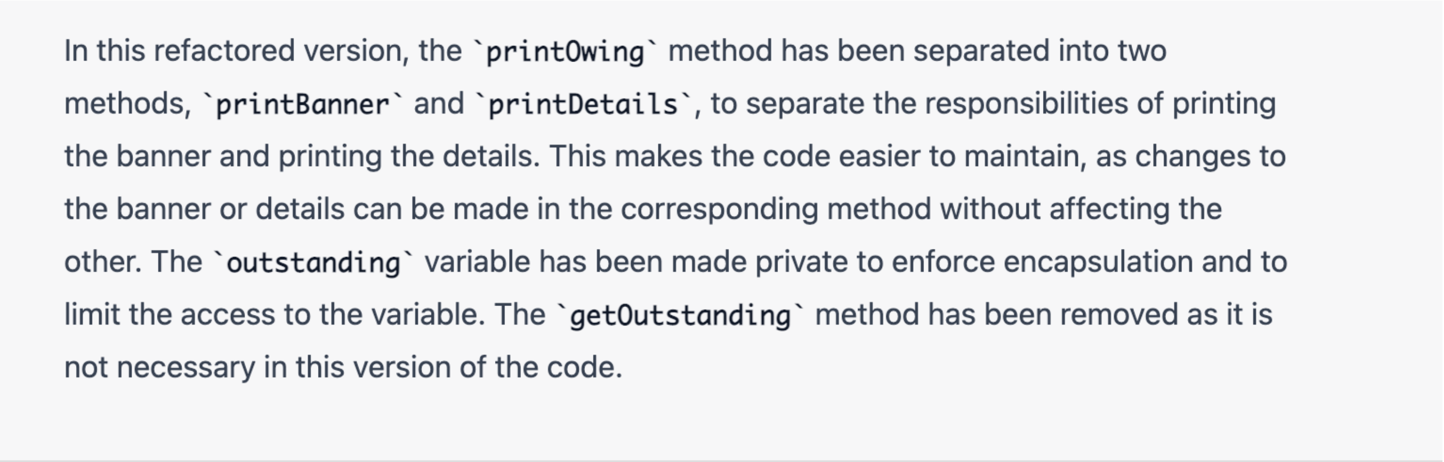
```
class Invoice {
  private String name;
  private double outstanding;

  public Invoice(String name, double outstanding) {
    this.name = name;
    this.outstanding = outstanding;
  }

  public void printOwing() {
    printBanner();
    printDetails();
  }

  private void printBanner() {
    // Print banner.
  }

  private void printDetails() {
    System.out.println("name: " + name);
    System.out.println("amount: " + outstanding);
  }
}
```



In this refactored version, the `printOwing` method has been separated into two methods, `printBanner` and `printDetails`, to separate the responsibilities of printing the banner and printing the details. This makes the code easier to maintain, as changes to the banner or details can be made in the corresponding method without affecting the other. The `outstanding` variable has been made private to enforce encapsulation and to limit the access to the variable. The `getOutstanding` method has been removed as it is not necessary in this version of the code.

Feel free to use but with

Can some metrics be used to aid refactoring?

Code Complexity

The ratio of time spent reading versus writing is well over 10 to 1

--Robert C Martin

- Code over time has tendency to accumulate complexity
- Greater or larger functionality should not have direct impact on code complexity
- Unnecessary complexity affects maintainability, time to market, understandability and testability

How to manage it? – Start measuring it!!

Definition from: Norman Fenton, Software Measurement: A Necessary Scientific Basis, IEEE TSE, March 1994

What is measurement?

Measurement is defined as the ***process*** *by which* ***numbers or symbols are assigned*** *to* ***attributes of entities*** *in the real world in such a way as to* ***describe them*** *according to clearly* ***defined rules***

Definition from: Norman Fenton, Software Measurement: A Necessary Scientific Basis, IEEE TSE, March 1994

What is measurement?

- Entity: can be an Object (person) or event (journey)
- Attribute: Feature of property of entity (height, blood pressure, etc.)
- Two types of measurement:
  - Direct measurement: measurement of attribute
  - Indirect measurement: Measurement of attribute involves measurement of some other attribute (eg: BMI)
- Uses of measurement – Assessment or Prediction

Measurement In terms of Software

- Carried out throughout the software development process

- Measurements can be performed at different levels
  - Completed Product (reliability, performance, etc.)
  - Development Process (time, man hours, etc.)
  - Source Code (lines of code, cyclomatic complexity, etc.)

- Source code metrics focus on measuring the source code of a system
  - Allows to measure complexity of code
    - Improve quality of code and thereby overall software
  - Used for lot of applications (defect prediction, fault localizations, refactoring, testing, etc.)

Commonly Used Source Code Metrics

- Lines of Code (LOC)
  - Easiest but effective indicator of complexity
  - Small modules have low defect rates as opposed to large ones

- Cyclomatic Complexity
  - Developed by Thomas McCabe, 1976
  - Allows to measure the complexity with respect to control flow of the code

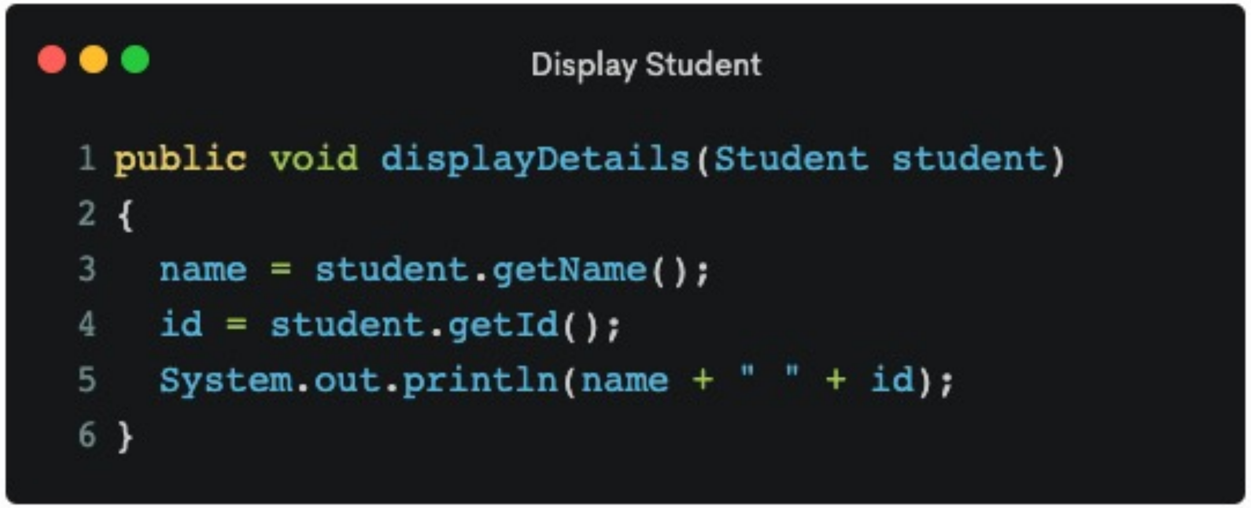
- Halstead Software Science Metrics
  - Developed by Halstead, 1977
  - Measures complexity in terms of the amount of information in source code

- There are also object oriented metrics (Chidamber and Kemerer 1994, Li and Henry 1993)

Cyclomatic Complexity

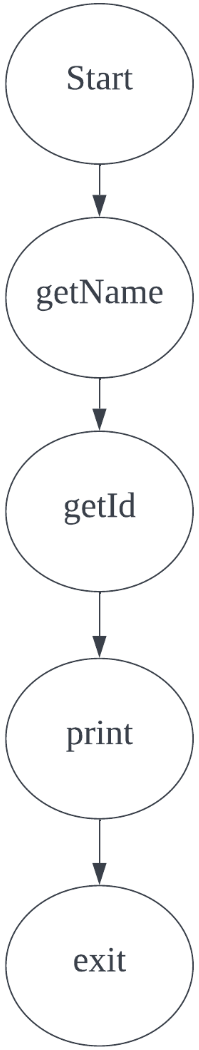
- Count of the number of linearly independent paths in a program
- Has a big impact on testing – test cases needs to cover the different paths
- Uses the control flow graph, G of the given program – Approach based on graph theory
- $V(G) = e - n + p$
  - e = Number of edges
  - n = Number of nodes
  - p = Connected components

Cyclomatic Complexity - Simple Example



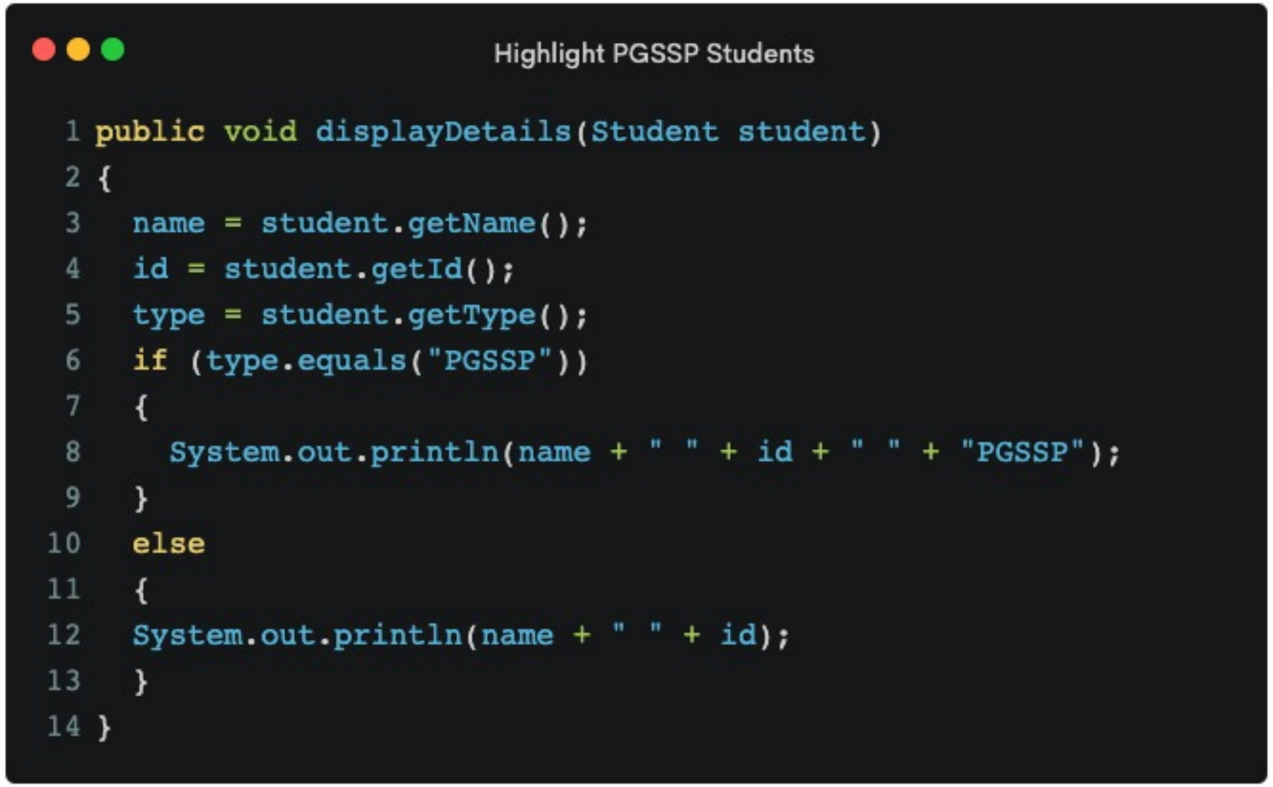
Display Student

```java
public void displayDetails(Student student)
{
  name = student.getName();
  id = student.getId();
  System.out.println(name + " " + id);
}
```



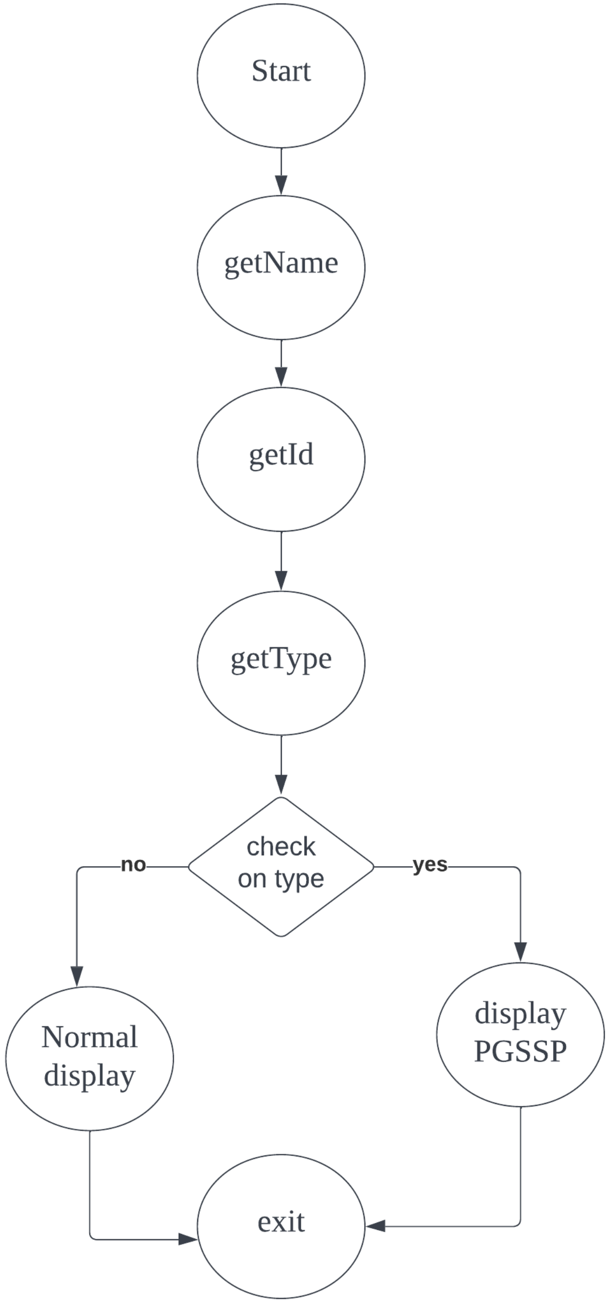
Complexity = 4 – 5 + 2*1
= 1

Cyclomatic Complexity - Another Example



Highlight PGSSP Students

```
public void displayDetails(Student student)
{
  name = student.getName();
  id = student.getId();
  type = student.getType();
  if (type.equals("PGSSP"))
  {
    System.out.println(name + " " + id + " " + "PGSSP");
  }
  else
  {
  System.out.println(name + " " + id);
  }
}
```



Complexity = 8 – 8 + 2*1

= 2

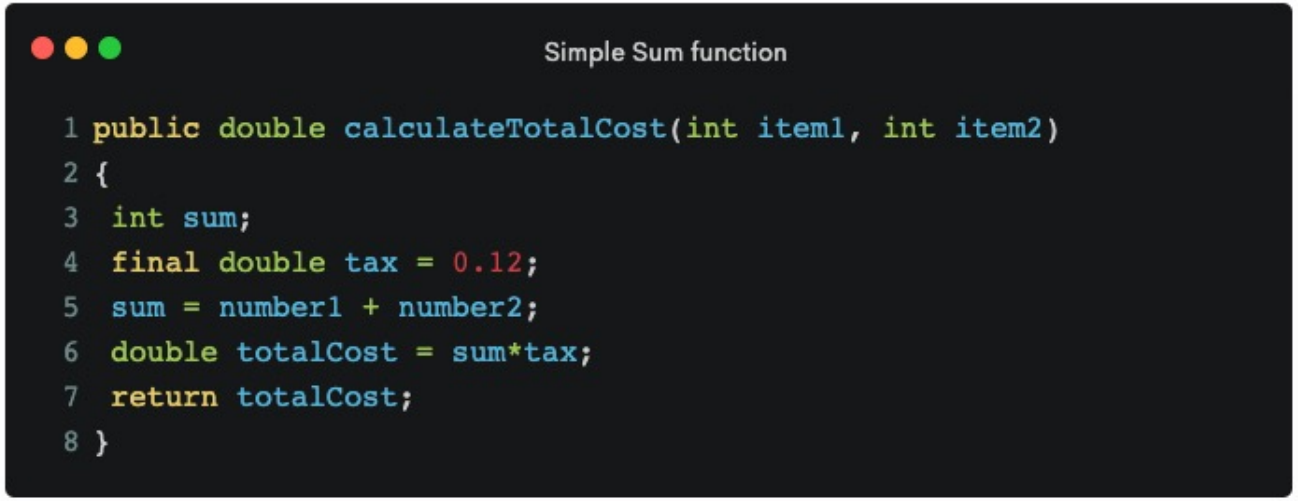
Halstead Software Science Metrics

- Considers program as a collection of tokens
- Tokens: Operators or operands
- The metrics makes use of the occurrence of operators and operands in a program to reason about complexity

n1 -> number of distinct operators (+, -, *, while, for, (), {}, function calls, etc.)
n2 -> number of distinct operands (variables, method names, etc.)
N1 -> total number of occurrence of operators
N2 -> total number of occurrence of operands

- The above observations are combined to provide different metrics

Halstead Software Science Metrics



```
public double calculateTotalCost(int item1, int item2)
{
 int sum;
 final double tax = 0.12;
 sum = number1 + number2;
 double totalCost = sum*tax;
 return totalCost;
}
```

- Vocabulary, n = n1 + n2
- Program length N = N1 + N2
- Volume, V = Nlog$_2$ (n)

….

Operators (+, *, =, double, int, final, return, {, }, (,)), n1 = **11**

Operands (calculateTotalCost, item1, item2, sum, tax, number1, number 2, totalCost) = **8**

N1 - (1, 1, 3, 3, 3, 1, 1,1,1,1,1) = **17**

n = 19, N = 28, V = 28log(19) = 35.80

N2 – (1, 1, 1, 2, 2, 1, 1, 2) = **11**

Six OO Metrics – Chidamber and Kemerer

- Weighted Methods per Class
- Depth of Inheritance Tree
- Number of Children of a Class
- Coupling Between Object Classes
- Response for a Class
- Lack of Cohesion on Methods

Image source: medium, google images

Thank You

Course website: karthikv1392.github.io/cs6401_se

Email: karthik.vaidhyanathan@iiit.ac.in
Web: https://karthikvaidhyanathan.com
Twitter: @karthi_ishere