Sugerencia de Correcciones de Código según los principios de diseño SOLID.

URL: https://github.com/justinliu23/student-information-system

LSP

En este caso se da una contravención al LISKOV SUBSTITUTION PRINCIPLE, ya que la clase padre Science posee un constructor sin argumentos y las dos clases hijas, Geology y Chemistry no lo poseen, por lo tanto, si se invoca las clases hijas con ese método constructor, se lanzaría un error. Esto va en contra ya que es un comportamiento no esperado.

```
package LSP1;
public class Science {
    //------------------------------
    // Represents the Science course
    //------------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public Science()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "S";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }
}
```

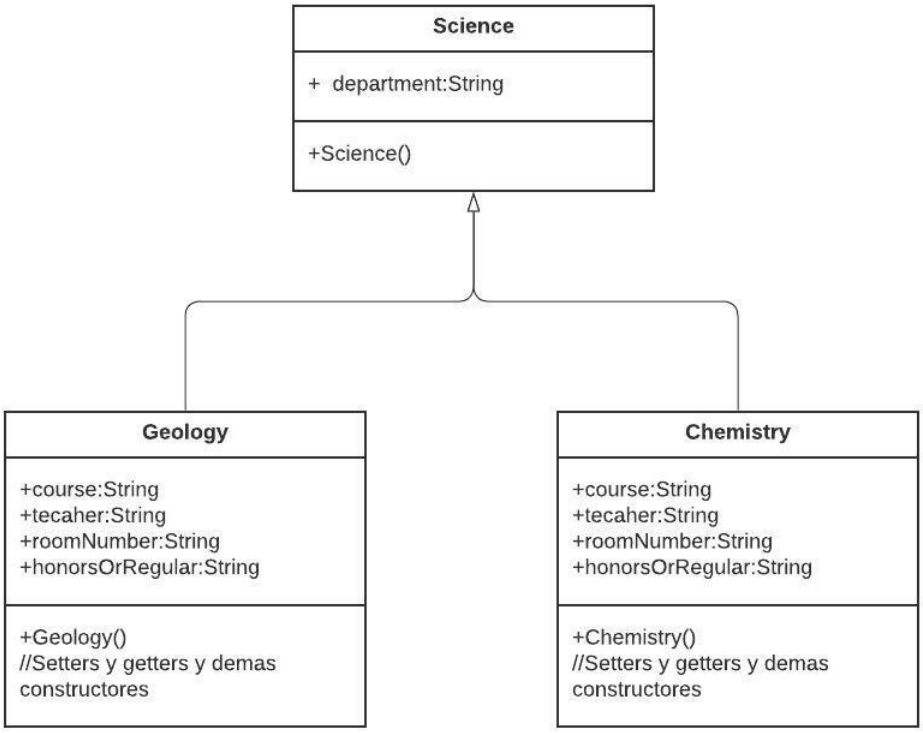
```
package LSP1;
public class Geology extends Science {

// Space to help line up and organize courses
String spacing;

public Geology (String honorsOrRegular)
{
    super();

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";
        // If the user chooses honors, decrease the spacing to counteract the extra spacing and misalignment
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";
        // If the user chooses regular, increase the spacing to compensate for the lessened spacing and misalignment
        this.spacing = "\t\t\t";
    }
}
```

```
package LSP1;
public class Chemistry extends Science {
```

```
String spacing;

public Chemistry (String honorsOrRegular)
{
    super();

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since chemistry contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since chemistry contains enough letters to account for space
        this.spacing = "\t\t";
    }
}

// Returns the course's title, then the teacher's last name, followed by the course's building and room number concatenated as a string
public String toString()
{
    return this.course + this.honorsOrRegular + this.spacing + this.teacher + "\t\t" + super.toString() + this.roomNumber + "\n";
}
```

SOLUCION

Se crea el constructor sin argumentos en las clases hija, con valores de acorde a la lógica del sistema en los atributos del hijo.



```
String spacing;

public Geology ()
{
    super();
    String honorsOrRegular="Honors";


    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";
        // If the user chooses honors, decrease the spacing to counteract the extra spacing and misalignment
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";
        // If the user chooses regular, increase the spacing to compensate for the lessened spacing and misalignment
        this.spacing = "\t\t\t";
    }
}
```

```
    String spacing;

    public Chemistry ()
    {
        super();
        String honorsOrRegular="Honors";

        // If the user wants to take honors, add an "H" at the end of the course title
        if (honorsOrRegular.equalsIgnoreCase("Honors"))
        {
            this.honorsOrRegular = " H";

            // Spacing will always be "\t\t" since chemistry contains enough letters to account for space
            this.spacing = "\t\t";
        }

        // Else leave the the course title alone
        else
        {
            this.honorsOrRegular = "";

            // Spacing will always be "\t\t" since chemistry contains enough letters to account for space
            this.spacing = "\t\t";
        }
    }
```

LSP:

En este caso se da una contravención al LISKOV SUBSTITUTION PRINCIPLE, ya que la clase padre Math posee un constructor sin argumentos y las tres clases hijas, Precalculus, Algebra2 y Geometry no lo poseen, por lo tanto, si se invoca las clases hijas con ese método constructor, se lanzaría un error. Esto va en contra ya que es un comportamiento no esperado.

```
package LSP2;

public class Math {
      //---------------------------
    // Represents the Math course
    //---------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public Math()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "M";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }

}
```

```
package LSP2;
public class Algebra2 extends Math {
```

```
public class Algebra2 extends Math {
```

```
String spacing;

public Algebra2 (String honorsOrRegular)
{
    super();

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since algebra 2 contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since algebra 2 contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

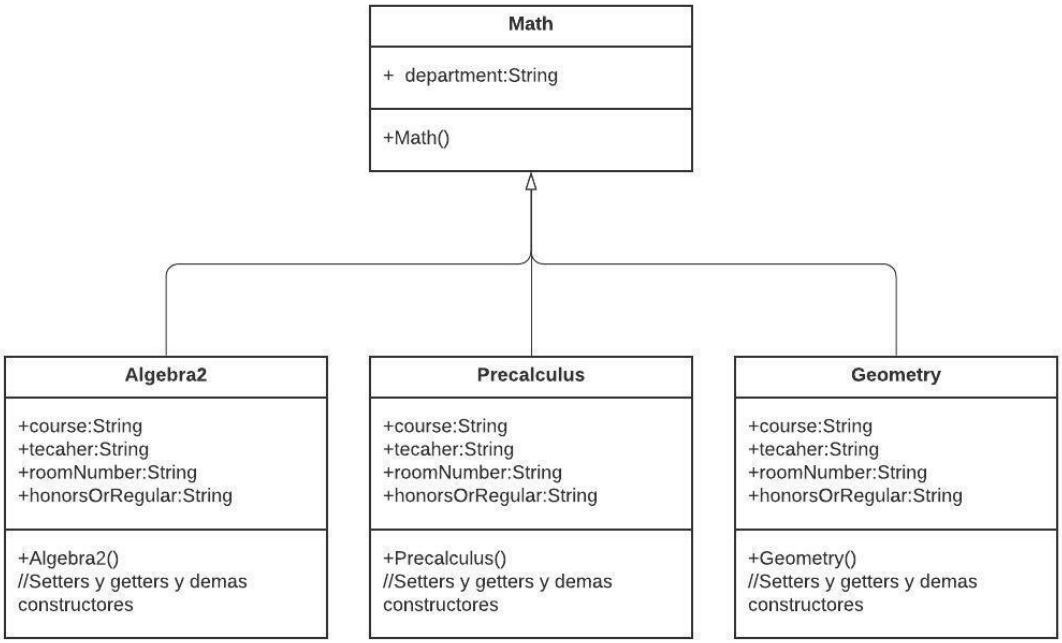
```
public class Geometry extends Math{
    //
```

```
String spacing;

public Geometry (String honorsOrRegular)
{
    super();

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since "geometry" contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since "geometry" contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

```
public class Precalculus extends Math {

String spacing;

public Precalculus (String honorsOrRegular)
{
    super();

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since "precalculus" contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since "precalculus" contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

Solución

Se crea el constructor sin argumentos en las clases hija, con valores de acorde a la lógica del sistema en los atributos del hijo.




```
String spacing;

public Precalculus ()
{
    super();
    String honorsOrRegular="Honors";

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since "precalculus" contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since "precalculus" contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

```
String spacing;

public Geometry ()
{
    super();
    String honorsOrRegular="Honors";

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since "geometry" contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since "geometry" contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

```
String spacing;

public Algebra2 ()
{
    super();
    String honorsOrRegular="Honors";

    // If the user wants to take honors, add an "H" at the end of the course title
    if (honorsOrRegular.equalsIgnoreCase("Honors"))
    {
        this.honorsOrRegular = " H";

        // Spacing will always be "\t\t" since algebra 2 contains enough letters to account for space
        this.spacing = "\t\t";
    }

    // Else leave the the course title alone
    else
    {
        this.honorsOrRegular = "";

        // Spacing will always be "\t\t" since algebra 2 contains enough letters to account for space
        this.spacing = "\t\t";
    }
}
```

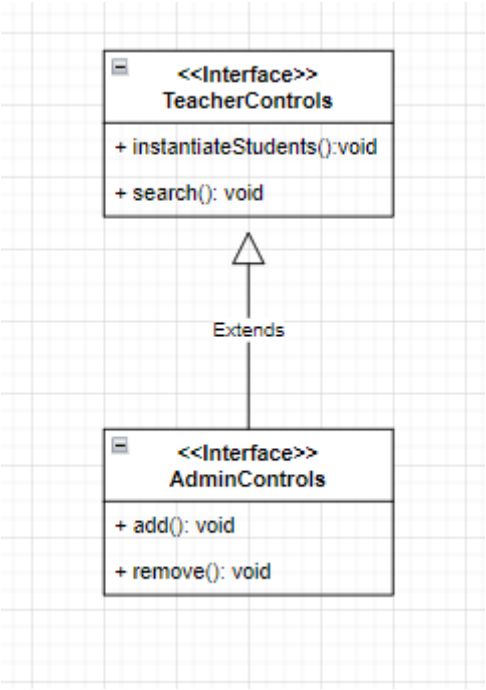
ISP:

```
public interface AdminControls {
    void instantiateStudents();
    void search();
    void add();
    void remove();
}
```

Solución

```
public interface AdminControls extends TeacherControls{
    void add();
    void remove();
}
public interface TeacherControls {
    void instantiateStudents();
    void search();
}
```

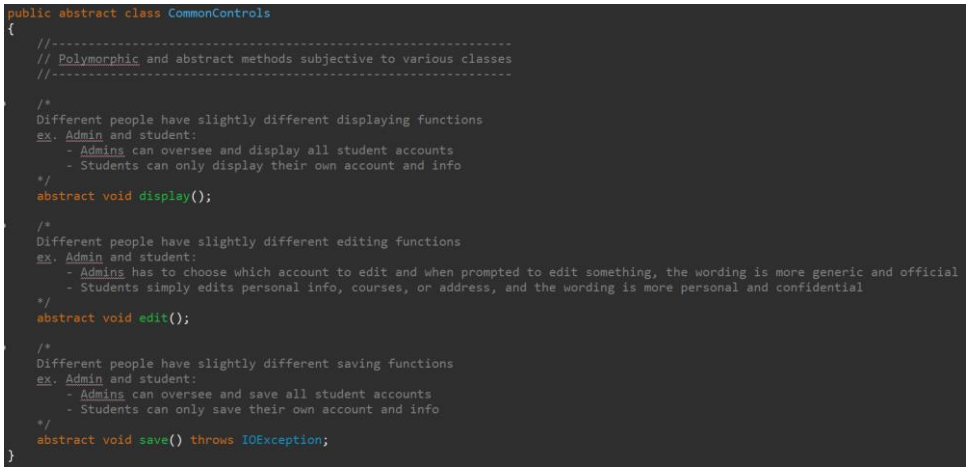
UML de la solución:



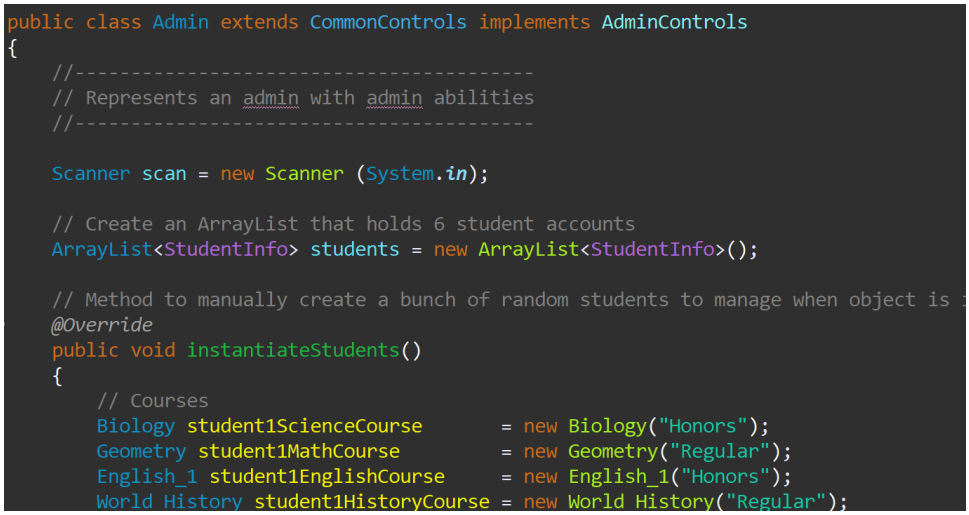
ISP:

Se preferiría que sea una interfaz en vez de clase abstracta

Esta clase abstracta tiene la finalidad de que sus métodos vayan orientados a la cuenta de los estudiantes, pero se debería segregar ya que siguiendo la violación ISP 2 se aspiraría que un profesor visualice la cuenta de los alumnos, pero no que las pueda editar ni guardar ya que no puede hacer cambios

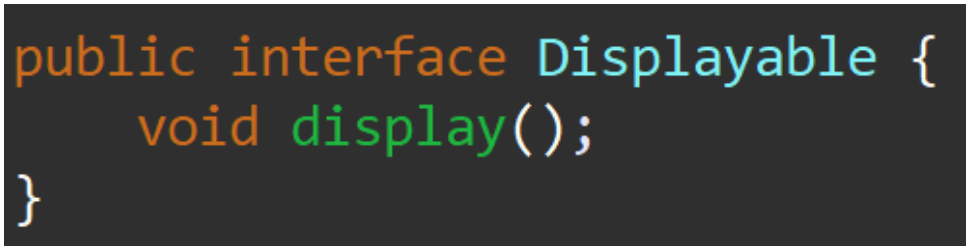


```
public abstract class CommonControls
{
    //---------------------------------------------------------------
    // Polymorphic and abstract methods subjective to various classes
    //---------------------------------------------------------------

    /*
    Different people have slightly different displaying functions
    ex. Admin and student:
        - Admins can oversee and display all student accounts
        - Students can only display their own account and info
    */
    abstract void display();

    /*
    Different people have slightly different editing functions
    ex. Admin and student:
        - Admins has to choose which account to edit and when prompted to edit something, the wording is more generic and official
        - Students simply edits personal info, courses, or address, and the wording is more personal and confidential
    */
    abstract void edit();

    /*
    Different people have slightly different saving functions
    ex. Admin and student:
        - Admins can oversee and save all student accounts
        - Students can only save their own account and info
    */
    abstract void save() throws IOException;
}
```



```
public class Admin extends CommonControls implements AdminControls
{
    //-----------------------------------------
    // Represents an admin with admin abilities
    //-----------------------------------------

    Scanner scan = new Scanner (System.in);

    // Create an ArrayList that holds 6 student accounts
    ArrayList<StudentInfo> students = new ArrayList<StudentInfo>();

    // Method to manually create a bunch of random students to manage when object is
    @Override
    public void instantiateStudents()
    {
        // Courses
        Biology student1ScienceCourse       = new Biology("Honors");
        Geometry student1MathCourse         = new Geometry("Regular");
        English_1 student1EnglishCourse     = new English_1("Honors");
        World_History student1HistoryCourse = new World_History("Regular");
```

Solución



```
public interface Displayable {
    void display();
}
```

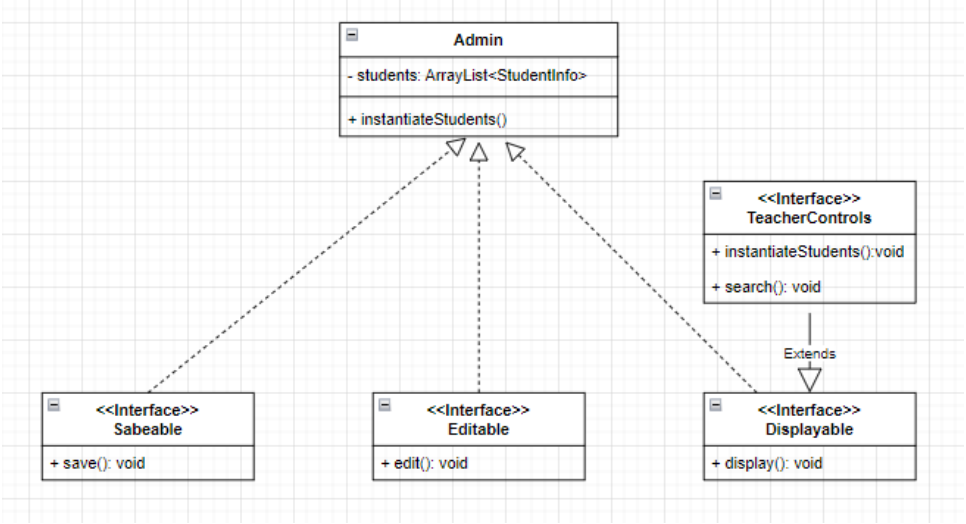
```java
public interface Editable {
    void edit();
}
```

```java
public interface Saveable {
    void save();
}
```

```java
public interface TeacherControls extends Displayable {
    void instantiateStudent();
    void search();
}
```

```java
public class Admin implements AdminControls, Displayable, Editable, Saveable
{
    //----------------------------------------
    // Represents an admin with admin abilities
    //----------------------------------------

    Scanner scan = new Scanner (System.in);

    // Create an ArrayList that holds 6 student accounts
    ArrayList<StudentInfo> students = new ArrayList<StudentInfo>();

    // Method to manually create a bunch of random students to manage when object is in
    @Override
    public void instantiateStudents()
    {
        // Courses
        Biology student1ScienceCourse       = new Biology("Honors");
        Geometry student1MathCourse         = new Geometry("Regular");
        English_1 student1EnglishCourse     = new English_1("Honors");
        World_History student1HistoryCourse = new World_History("Regular");
```

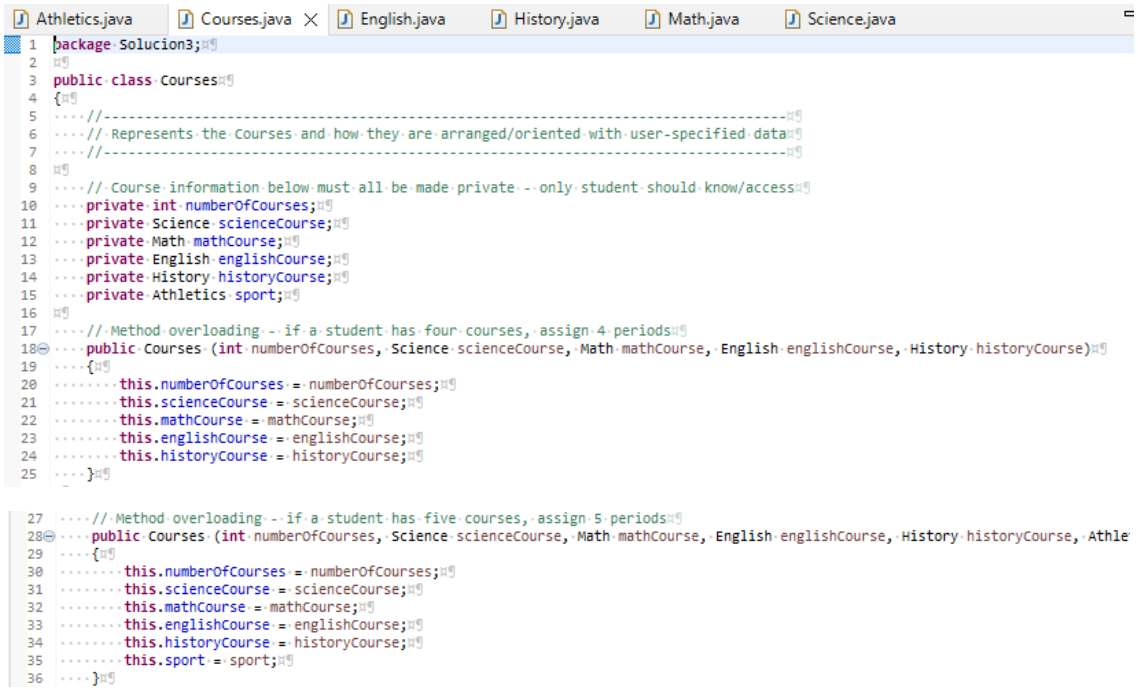
UML de la solución:



DIP:

La clase Courses depende de clases de bajo nivel, entre ellas están las clases English, History, Athletics, Math y Science, lo que viola el quinto principio SOLID.

Esto hace inconsistente a esta clase porque tiene una repercusión en otras clases, esto quiere decir que si se modifica una de esas clases será difícil llevar a cabo el cambio. Esto se lo puede apreciar porque los parámetros de los dos constructores requieren de dichas variables que se están referenciando de otras clases.



Solución:

La clase Courses deja de depender de clases de bajo nivel porque ahora depende de abstracciones tanto en la inicialización de variables como en los parámetros del constructor de la clase Courses.

```java
package Solucion3;

public class Courses
{
    //-----------------------------------------------------------------------
    // Represents the Courses and how they are arranged/oriented with user-specified data
    //-----------------------------------------------------------------------

    // Course information below must all be made private - only student should know/access
    private int numberOfCourses;
    private Sciencable scienceCourse;
    private Matheable mathCourse;
    private Englishable englishCourse;
    private Historyable historyCourse;
    private Athleticable sport;

    // Method overloading - if a student has four courses, assign 4 periods
    public Courses (int numberOfCourses, Sciencable scienceCourse, Matheable mathCourse, Englishable englishCourse, Historyable historyCourse)
    {
        this.numberOfCourses = numberOfCourses;
        this.scienceCourse = scienceCourse;
        this.mathCourse = mathCourse;
        this.englishCourse = englishCourse;
        this.historyCourse = historyCourse;
    }

    // Method overloading - if a student has five courses, assign 5 periods
    public Courses (int numberOfCourses, Sciencable scienceCourse, Matheable mathCourse, Englishable englishCourse, Historyable historyCourse, Athleticable sport)
    {
        this.numberOfCourses = numberOfCourses;
        this.scienceCourse = scienceCourse;
        this.mathCourse = mathCourse;
        this.englishCourse = englishCourse;
        this.historyCourse = historyCourse;
        this.sport = sport;
    }

```

Esto se logra gracias a que las clases de bajo nivel que antes dependía la clase Courses, se les creara una interfaz que pueda depender de ella también. Las clases implementan estas interfaces y así se logra la no violación del principio DIP.

```java
package Solucion3;

public class English implements Englishable
{
    //------------------------------
    // Represents the English course
    //------------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public English()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "E";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }
}
```

```java
package Solucion3;

public interface Englishable {

    public String toString();
}
```

```java
package Solucion3;

public class English implements Englishable
{
    //------------------------------
    // Represents the English course
    //------------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public English()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "E";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }
}
```

```java
package Solucion3;

public interface Historyable {

    public String toString();
}
```

```java
package Solucion3;

public class Math implements Matheable
{
    //------------------------------
    // Represents the Math course
    //------------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public Math()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "M";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }
}
```

```java
package Solucion3;

public interface Matheable {

    public String toString();
}
```

```java
package Solucion3;

public class Science implements Sciencable
{
    //------------------------------
    // Represents the Science course
    //------------------------------

    // Department information should be known/accessible to everyone
    public String department;

    public Science()
    {
        // Determines department and also identifies the department in which the specific field is located
        this.department = "S";
    }

    // Returns the department of this course as a String
    public String toString()
    {
        return this.department;
    }
}
```

```java
package Solucion3;

public interface Sciencable {
    public String toString();

}
```

UML de la solución