A) Bugs

1)Line 244:

**LoadingWidget.getInstance().hide();**

**iDetails = result;**

**iContext.setStudentId(iDetails == null ? null : iDetails.getStudentId());**

**iSpecRegCx.setCanRequire(iDetails == null || iDetails.isCanRequire());**

**header.setEnabled("submit", result.isCanUpdate());**

Description:

**A "NullPointerException" could be thrown; "result" is nullable here.**

**Implies 'iDetails' can be null and 'result' is dereferenced.** **A reference to null should never be accessed. Doing so will cause a NullPointerException to be thrown. At best, such an exception will cause abrupt program termination. At worst, it could expose debugging information that would be useful to an attacker, or it could allow an attacker to bypass security measures.**

B)Code Smells

1) Line 128

**private UniTimeHeaderPanel header, footer;**

Description:

**Declare "footer" on a separate line. Declaring multiple variables on one line is difficult to read.**

2)Line 167

**public AdvisorCourseRequestsPage() {---}**

Description:

**Refactor this method to reduce its Cognitive Complexity from 198 to the 15 allowed. Cognitive Complexity is a measure of how hard the control flow of a method is to understand. Methods with high Cognitive Complexity will be difficult to maintain. Exceptions equals and hashCode methods are ignored because they might be automatically generated and might end up being difficult to understand, especially in presence of many fields.**

3)Line 256

**switch (result.getWaitListMode()) {----}**

Description:

**Replace this "switch" statement by "if" statements to increase readability. switch statements are useful when there are many different cases depending on the value of the same expression.For just one or two cases however, the code will be more readable with if statements.**

4)Line 418

**iCourses = new ArrayList<AdvisorCourseRequestLine>();**

Description:

**Replace the type specification in this constructor call with the diamond operator ("<>"). Java 7 introduced the diamond operator (<>) to reduce the verbosity of generics code. For instance, instead of having to declare a List's type in both its declaration and its constructor, you can now simplify the constructor declaration with <>, and the compiler will infer the type.**

5)Line 643:

**if (session != null && !matchSession(info, session)) return false;**

Description:

**Replace this if-then-else statement by a single return statement. Return of boolean literal statements wrapped into if-then-else ones should be simplified.Similarly, method invocations wrapped into if-then-else differing only from boolean literals should be simplified into a single invocation.**

6)Line 648

**public void onFailure(Throwable caught) {}**

Description:

**Add a nested comment explaining why this method is empty, throw an UnsupportedOperationException or complete the implementation. There are several reasons for a method not to have a method body.First,It is an unintentional omission, and should be fixed to prevent an unexpected behavior in production.Second,It is not yet, or never will be, supported. In this case an UnsupportedOperationException should be thrown.Third,The method is an intentionally-blank override. In this case a nested comment should explain the reason for the blank override.**

7)Line1022

**while (iAlternatives.size() < request.getAlternatives().size()) addAlternativeLine();;**

Description:

**This statement will not be executed in a loop; only the first statement will be. The rest will execute only once. Curly braces can be omitted from a one-line block, such as with an if statement or for loop, but doing so can be misleading and induce bugs.This rule raises an issue when the whitespacing of the lines after a one line block indicates an intent to include those lines in the block, but the omission of curly braces means the lines will be unconditionally executed once.**

8) Line1170

**if (accept && it.hasNext()) {**

Description:

**When boxed type java.lang.Boolean is used as an expression to determine the control flow .it will throw a NullPointerException if the value is null.It is safer to avoid such conversion altogether and handle the null value explicitly.**

9)Line1333

**if (prefs != null) hasPref = true;WebTable.Cell credit = new WebTable.Cell(rc.hasCredit()?(rc.getCreditMin().equals(rc.getCreditMax()) ?df.format(rc.getCreditMin()) : df.format(rc.getCreditMin()) + " - " +df.format(rc.getCreditMax())) : "");**

Description:

**Extract this nested ternary operation into an independent statement. Just because you can do something, doesn’t mean you should, and that’s the case with nested ternary operations. Nesting ternary operators results in the kind of code that may seem clear as day when you write it, but six months later will leave maintainers (or worse - future you) scratching their heads and cursing.Instead, err on the side of clarity, and use another line to express the nested operation as a separate statement.**

10)Line1362

**String free = "";**

**for (FreeTime ft: rc.getFreeTime()) {**

**if (!free.isEmpty()) free += ", ";**

Description:

**Use a StringBuilder instead. Strings are immutable objects, so concatenation doesn’t simply add the new String to the end of the existing string. Instead, in each loop iteration, the first String is converted to an intermediate object type, the second string is appended, and then the intermediate object is converted back to a String. Further, performance of these intermediate operations degrades as the String gets longer. Therefore, the use of StringBuilder is preferred.**

11)Line1667

**public static enum Column {**

Description:

**Remove this redundant "static" qualifier; nested enum types are implicitly static. Nested enum types are implicitly static.So there’s no need to declare them static explicitly.**

Note There are other code smells that are repeated so I didn't write it.