UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
CURSO DE GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

ATHOS LAGEMANN
CAMILA PRIMIERI
GABRIEL CONTE
GUNTER HERTZ

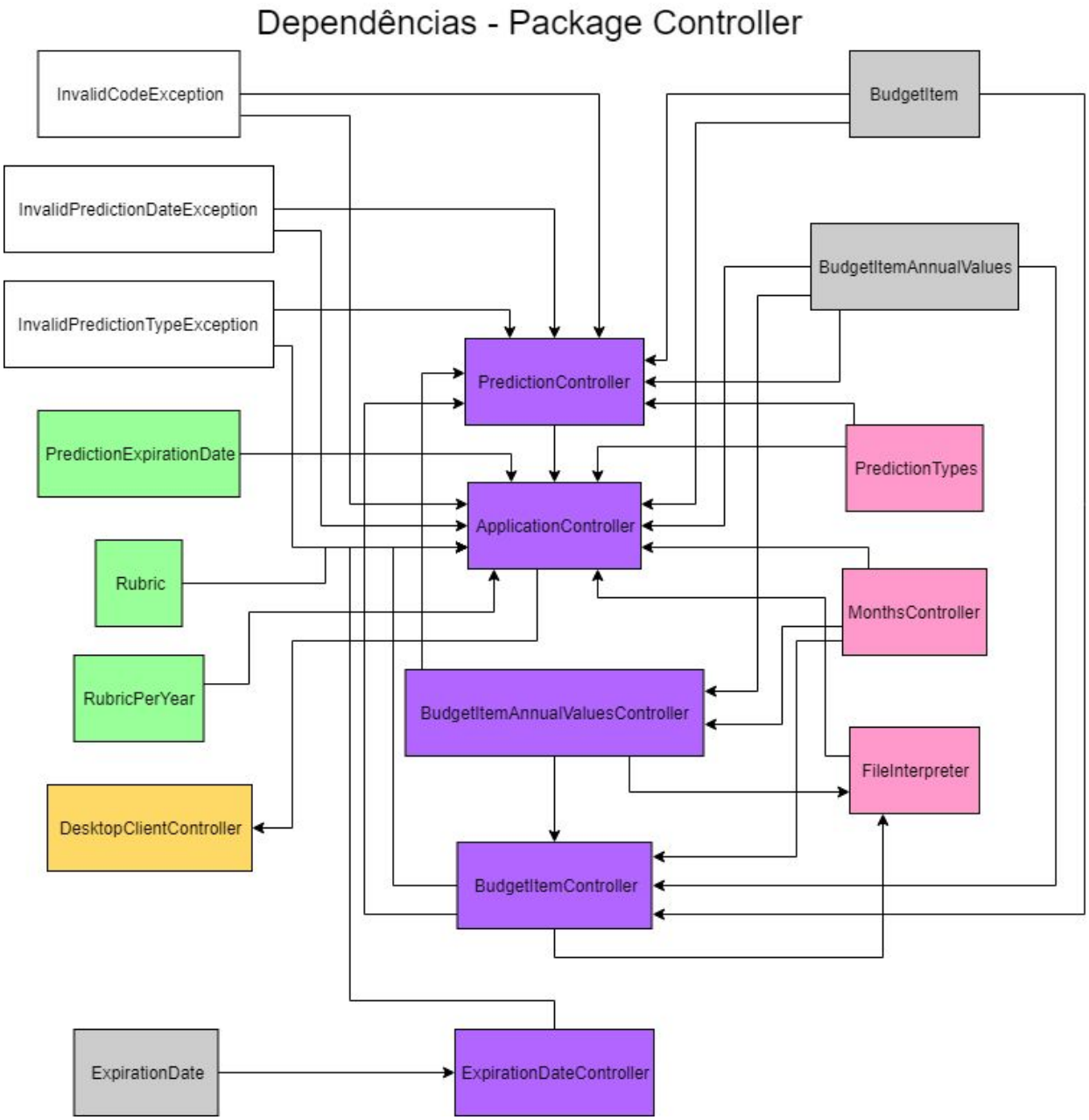
Relatório de Testes de Software

Orientadora: Prof. Erika Costa

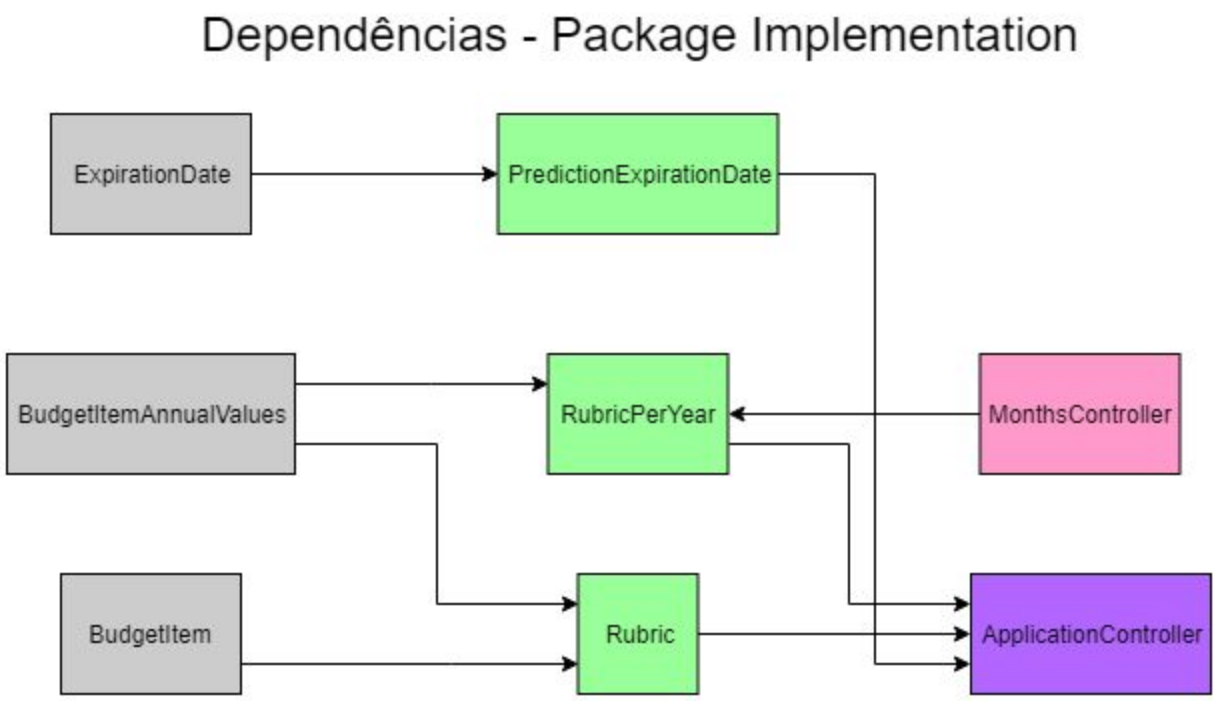
Porto Alegre
2018

Grafos de Dependência das Classes

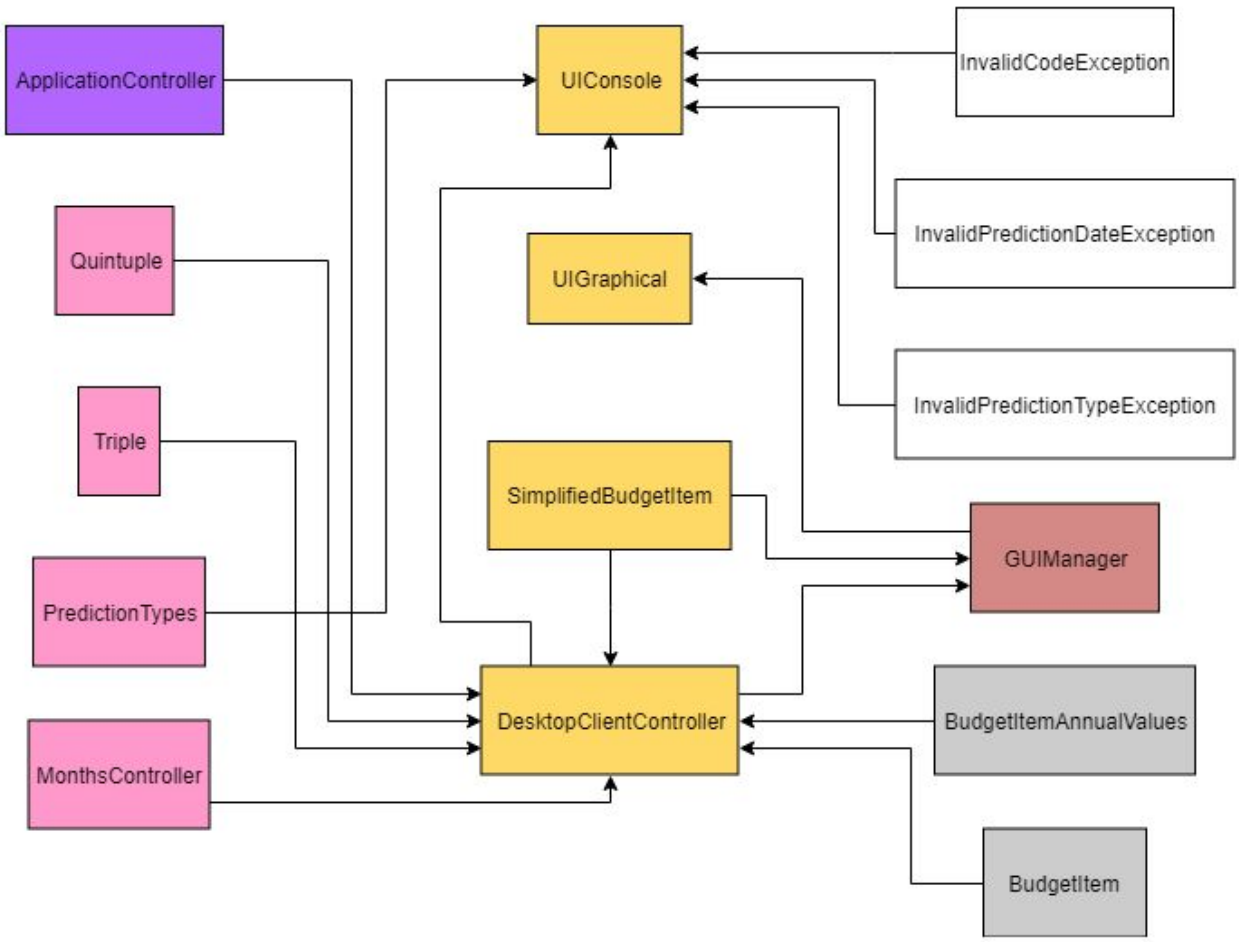
Iniciamos o trabalho gerando o grafo de dependências das classes do sistema. Geramos um grafo único que representava todo o sistema, mas, devido a complexidade das dependências, acabamos por desmembrá-lo em um grafo para cada pacote de classes.



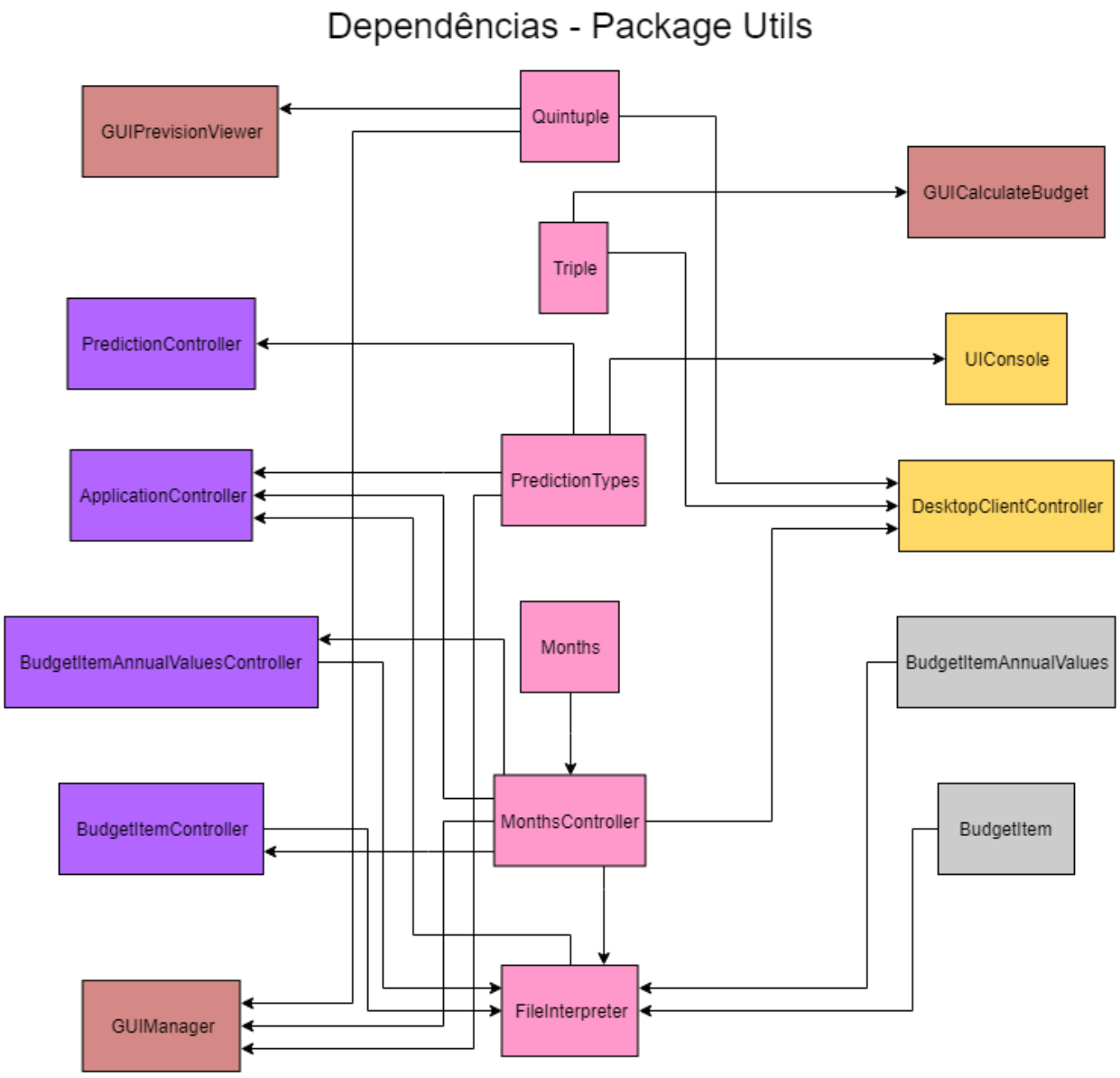
O pacote Controller possui diversas dependências, principalmente a classe ApplicationController, que depende de outras 14 classes. Nenhuma das classes do Controller são consideradas folhas no grafo de dependências.



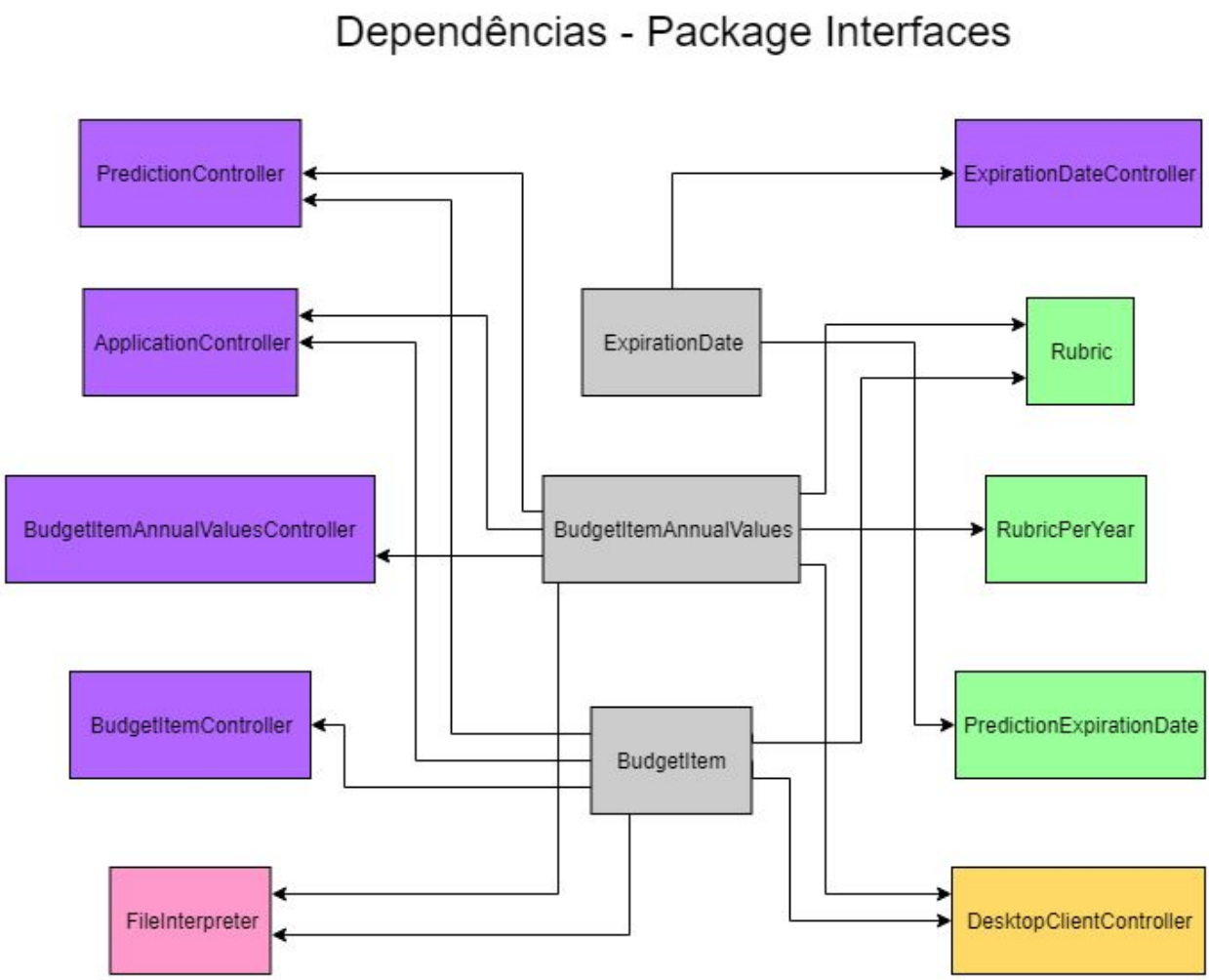
O pacote Implementation possui apenas 3 classes: Rubric, RubricPerYear e PredictionExpirationDate. Elas dependem principalmente das interfaces ExpirationDate, BudgetItemAnnualValues e BudgetItem, além da classe MonthsController, que se encontra no pacote Utils. E são todas elas dependências da classe ApplicationController.



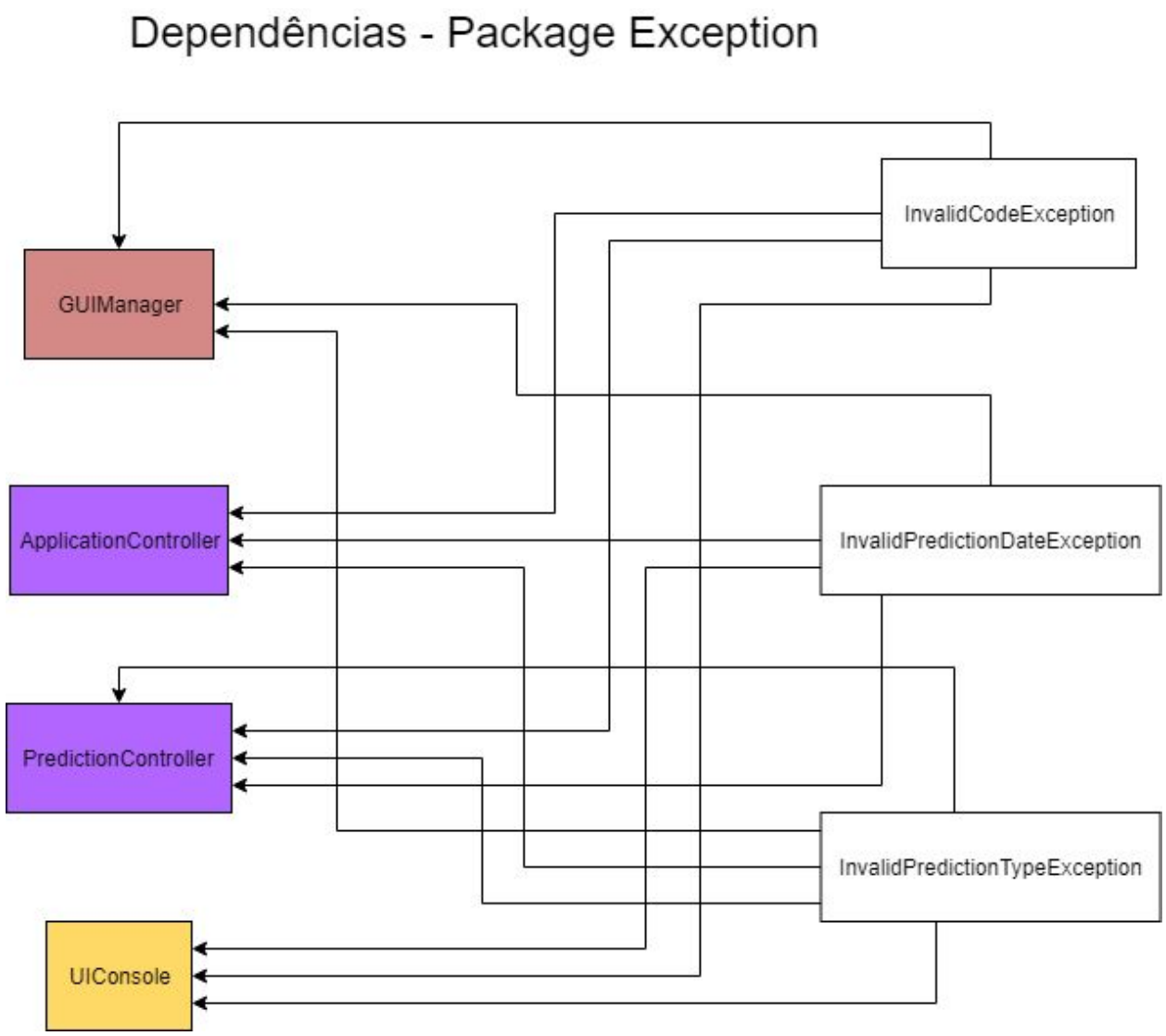
O pacote Interaction possui uma classe que é folha na árvore de dependência: SimplifiedBudgetItem, enquanto as demais dependem principalmente do pacote Utils, Interfaces e Exceptions. A classe UIGraphical depende somente da classe GUIManager, porém esta possui diversas dependências, como demonstra o grafo do pacote GUI.



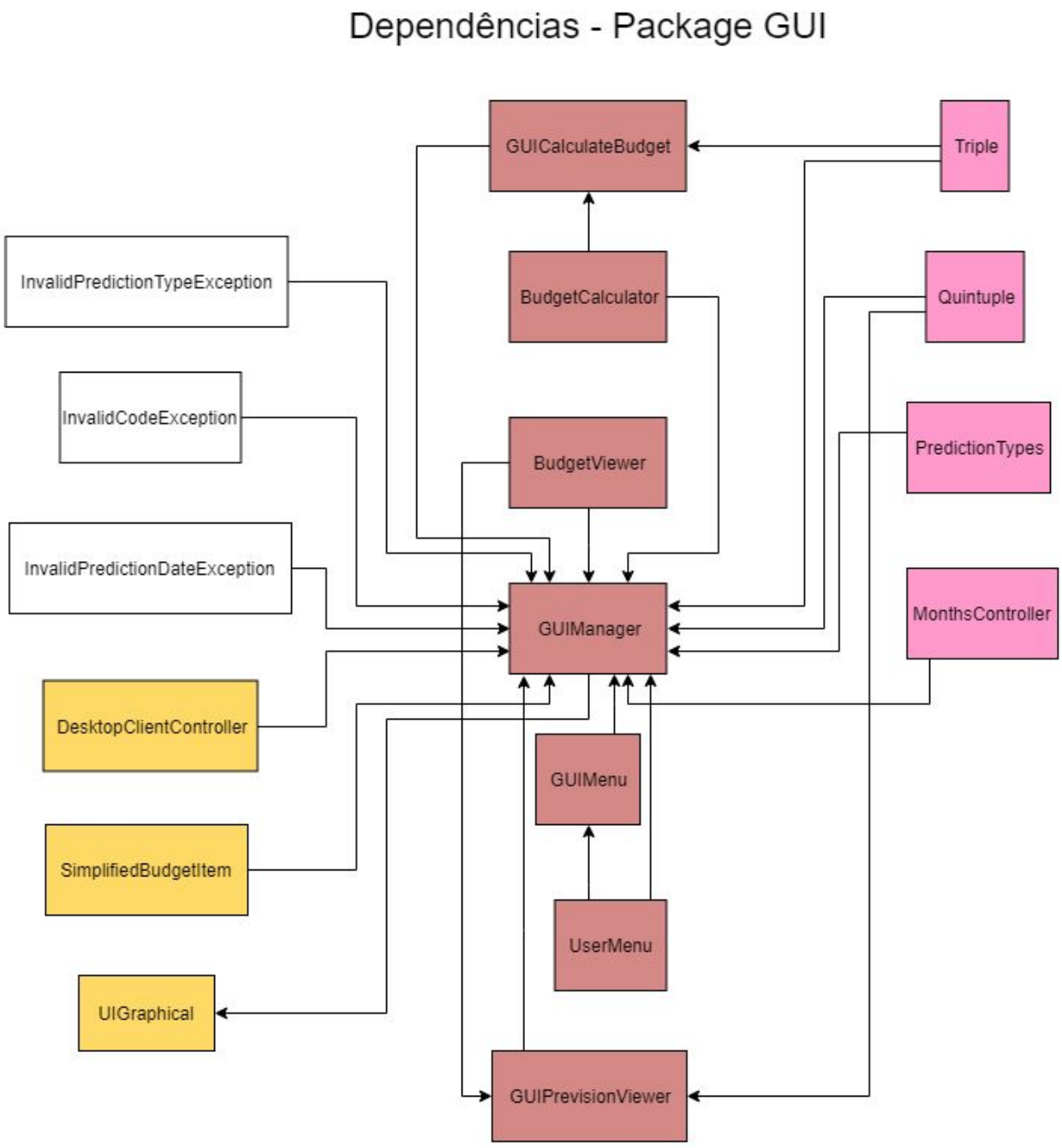
O pacote Utils possui diversas classes que são folha no grafo de dependência: Quintuple, Triple, Prediction Types e Months. Estas classes são utilizadas para diversas outras, principalmente do pacote Controller e GUI. Já a classe FileInterpreter possui diversas dependências e foi teve seus métodos testados e representados na forma de grafos CFG.



O pacote Interfaces é composto apenas de classes que são folhas no grafo de dependências. Delas depende principalmente classes dos pacotes Controllers e Implementation, além do FileInterpreter do pacote Utils e DesktopClientController do pacote Interaction.



O pacote Exception é composto apenas de nós-folha do grafo de dependências de classes. Todas as classes implementam as exceções disparadas em caso de erros conhecidos e tratados do sistema, sendo chamadas principalmente pelas classes do pacote Controller. Além disso, as classes GUIManager e UIConsole também utilizam as classes de exceções.

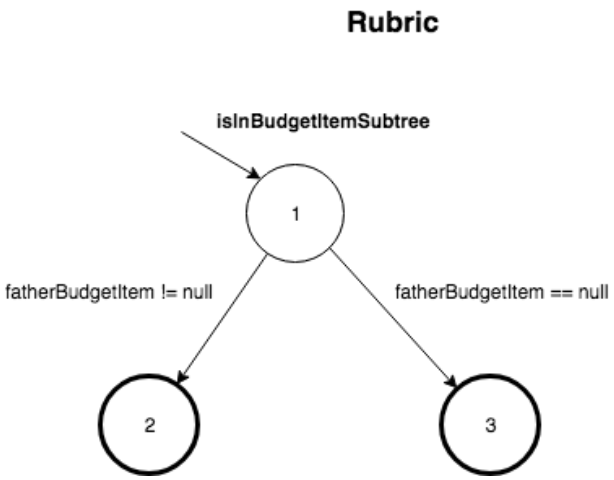


Por fim, há o grafo de dependências das classes do pacote GUI, que trata da interface gráfica do sistema. A classe GUIManager é que possui o maior número de dependências, enquanto as classes UserMenu, BudgetCalculator e BudgetViewer são nós-folha da árvore de dependências das classes. As classes dos pacotes Exception, Interaction e Utils são as que possuem dependência ou das quais as classes do pacote GUI dependem.

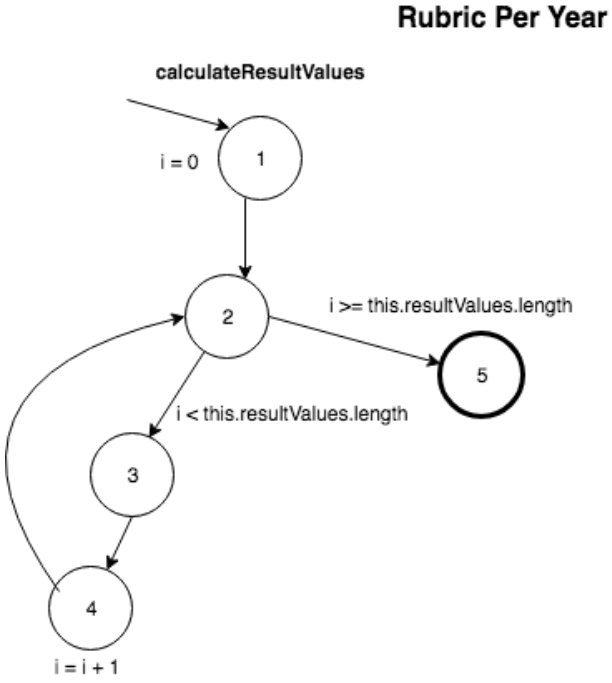
Grafos de Controle de Fluxo

Uma vez feito e analisado os grafos de dependências de classes, foram desenhados os grafos de controle de fluxo dos módulos das classes folha. Os CFG foram de grande importância para entender o fluxo dos módulos das classes e definir os locais onde foram necessários ser feitos testes unitários. Abaixo seguem o CFG's gerados e suas classes correspondentes:

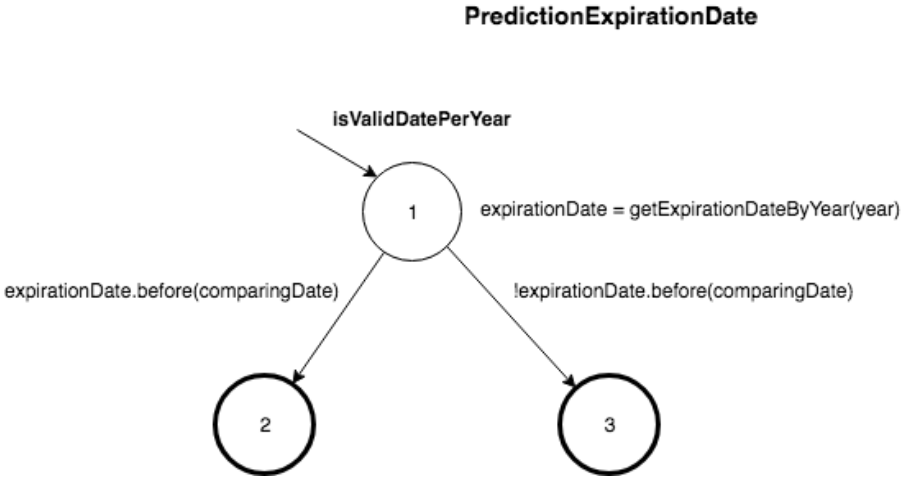
- **Classe Rubric**



- **Classe RubricPerYear**

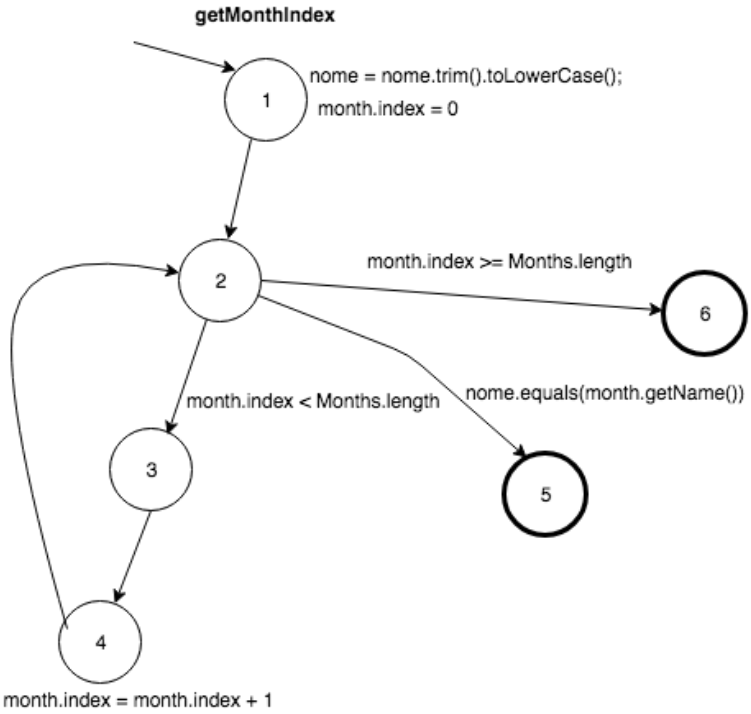


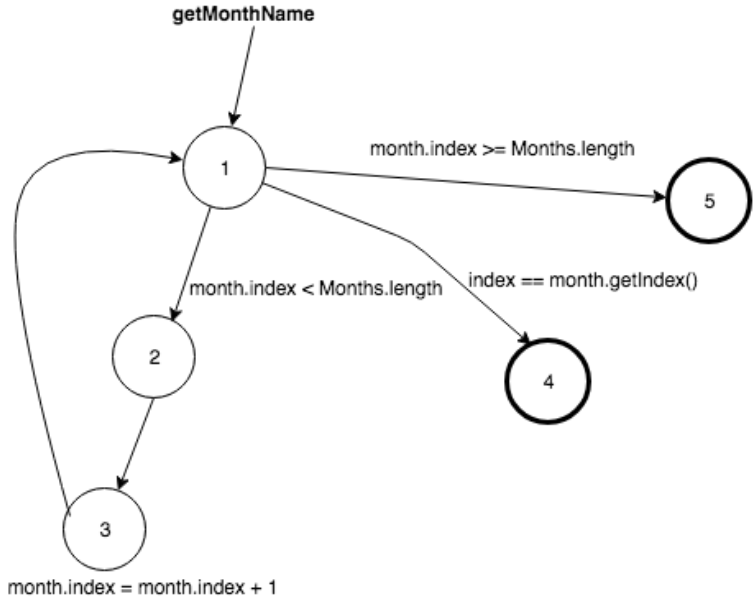
- **Classe PredictionExpirationDate**



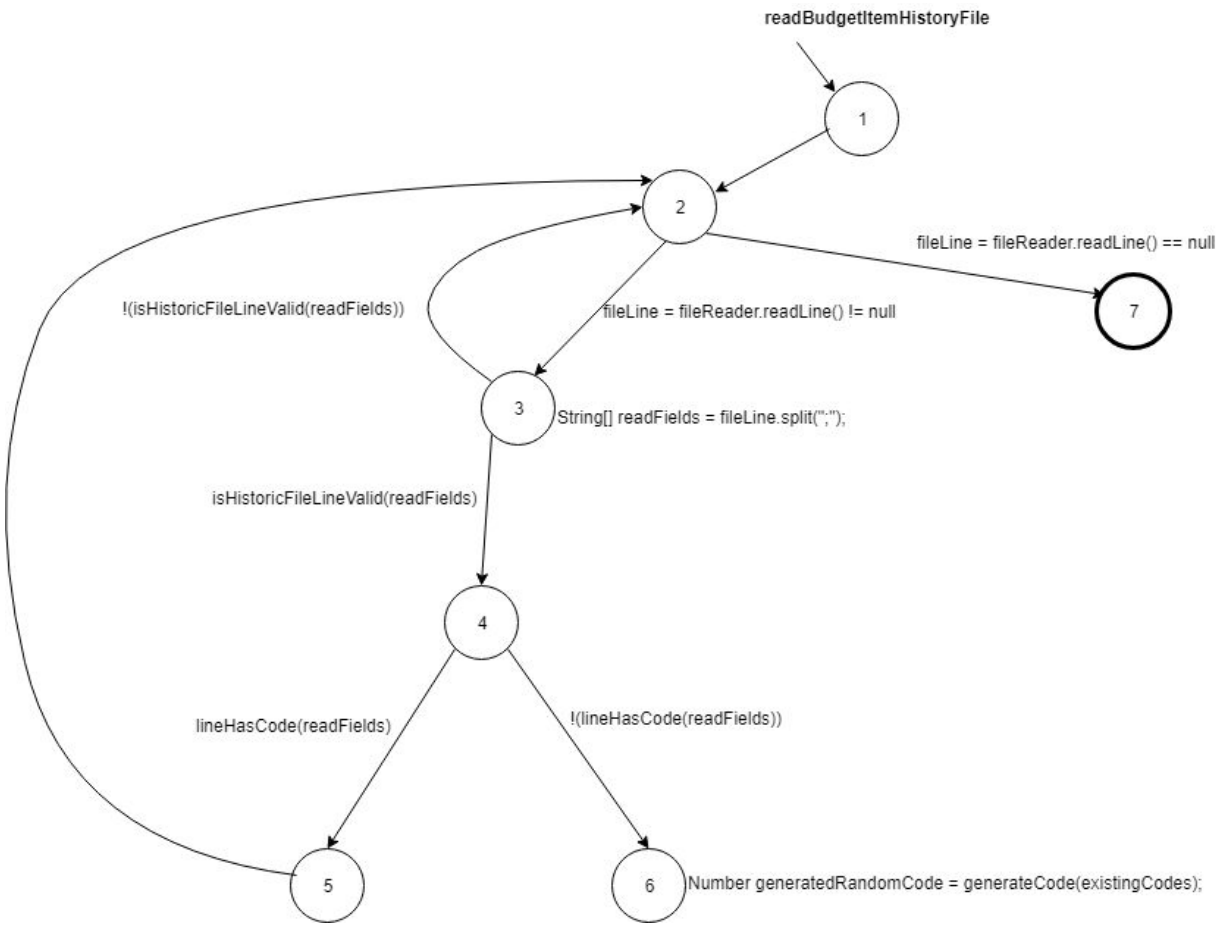
- **Classe MonthController**

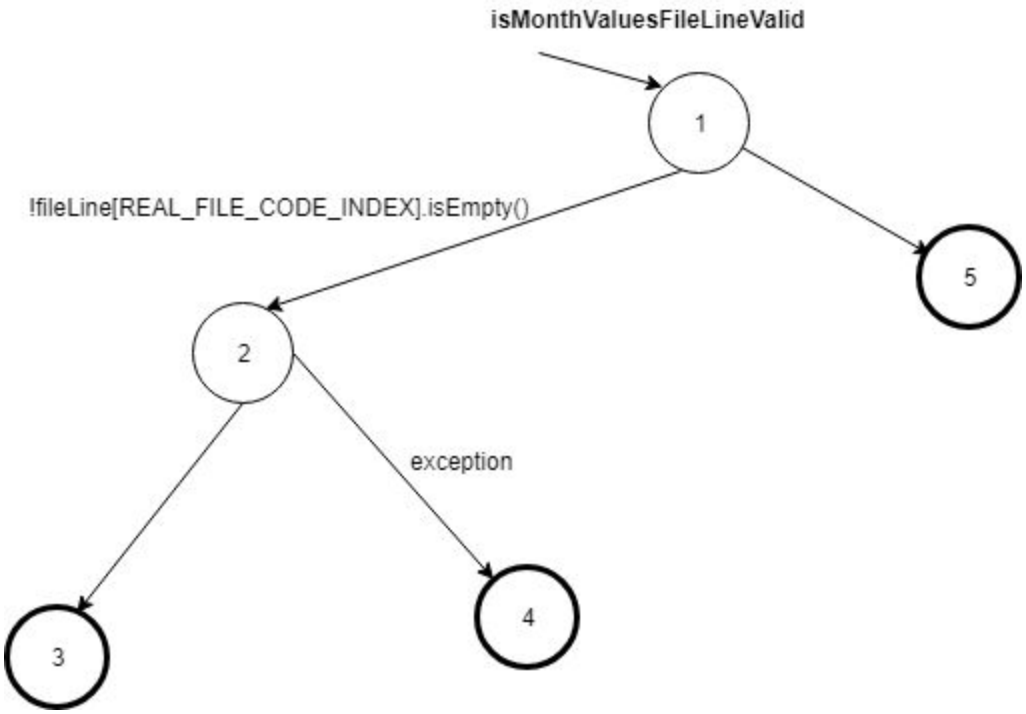
Month Controller



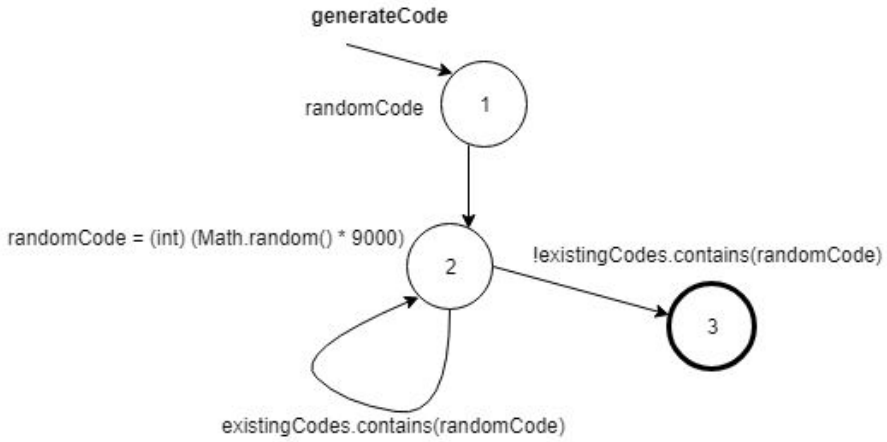


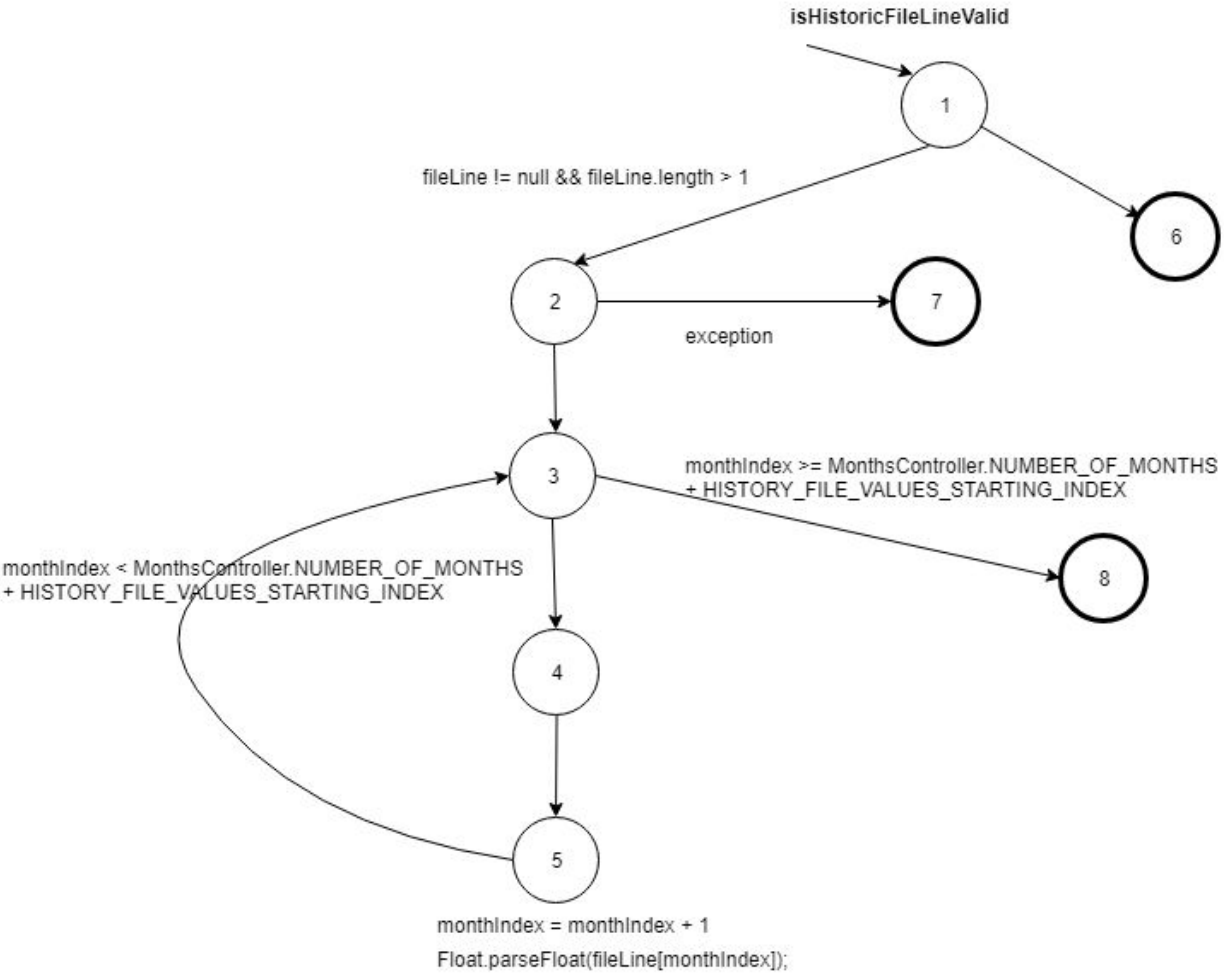
- **Classe FileInterpreter**

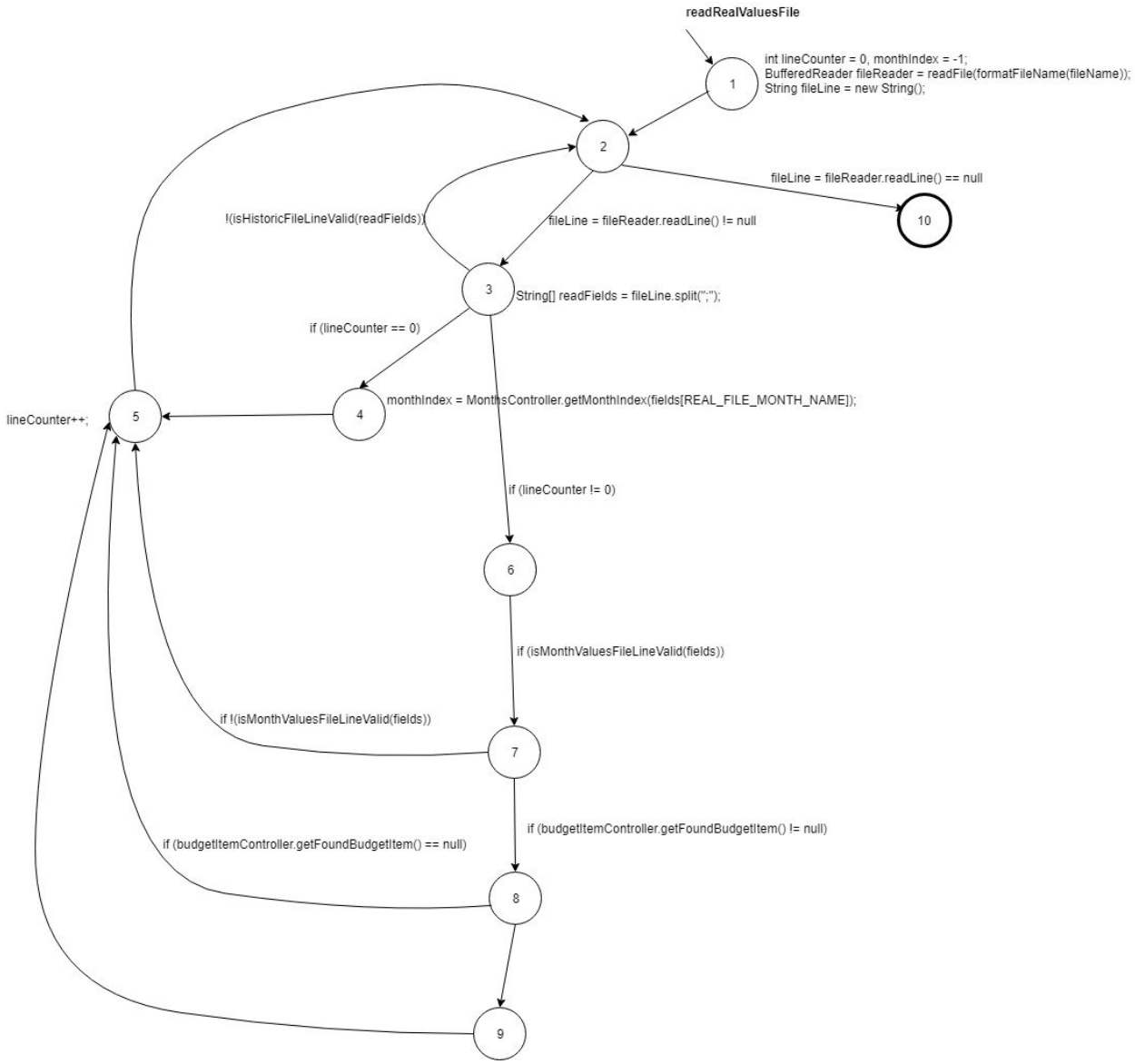



isMonthValuesFileLineValid
1
!fileLine[REAL_FILE_CODE_INDEX].isEmpty()
5
2
exception
4
3

File Interpreter






readRealValuesFile
1
int lineCounter = 0, monthIndex = -1;
BufferedReader fileReader = readFile(formatFileName(fileName));
String fileLine = new String();
2
fileLine = fileReader.readLine() == null
10
!(isHistoricFileLineValid(readFields))
fileLine = fileReader.readLine() != null
3
String[] readFields = fileLine.split(";");
if (lineCounter == 0)
4
monthIndex = MonthsController.getMonthIndex(fields[REAL_FILE_MONTH_NAME]);
lineCounter++;
5
if (lineCounter != 0)
6
if (isMonthValuesFileLineValid(fields))
if !(isMonthValuesFileLineValid(fields))
7
if (budgetItemController.getFoundBudgetItem() != null)
if (budgetItemController.getFoundBudgetItem() == null)
8
9

Documentação Testes unitários

Foram feitos testes unitários em algumas classes que acreditávamos ser as folhas, porém elas representavam os modelos existentes no projeto e somente possuíam construtor e métodos de getters e setters, a lógica da aplicação não estava nela. A partir disso foi feito os grafos de dependência para levantar quais seriam as classes folhas e que deveríamos realizar os testes.

O objetivo dos testes criados era realizar validações simples, por exemplo: se um objeto estaria nulo, se o número de elementos em um array corresponde ao número de elementos adicionados a ele, etc. Os testes foram feitos nas seguintes classes: FileInterpreter, MonthsController, Rubric, RubricPerYear, PredictionExpirationDate, e a cobertura de testes do projeto conseguiu avançar para 39%.

Modificacoes:

- *Rubric:*
    - Realizado teste no construtor: testa se o Array e o Hashmap não são nulos.
    - Realizado teste nos métodos de getters e setters.

- *ExpirationDateController*
    - Retirado assert != null do método getExpirationDateByYear e iSValidDatePerYear.

- *RubricPerYear*:
    - Testamos se getResultValues.lenght == valor inicializado na classe.

- *FileInterpreter*:
    - Adicionados testes a outras planilhas .csv, com estruturas diferentes. Assim, os caminhos interpretados eram diferentes, cobrindo uma gama de possibilidades maior.

- *MonthsController* (nova classe):
    - Adicionamos testes para cada mês possível, de 0 a 13, para incluir o caso inválido.