feup-mfes

January 6, 2019

Contents

1 Document **1**

2 Employee **3**

3 Malfunction **4**

4 Printer **5**

5 PrintingQueue **9**

6 Service **10**

7 Student **12**

8 University **15**

9 EmployeeTests **16**

10 MalfunctionTests **16**

11 PrinterTests **18**

12 StudentTests **20**

13 TestSuite **22**

1 Document

```
/**
 * A document created by a student with the end purpose of being printed.
 */
class Document

types
 /** Type definition for string. */
 public String = seq of char;

 /** A document is saved with the option of being printed either in black and white or in color.
     */
 public Color = <BW>|<COLOR>;
```

```
 /** A document is saved with either A3 or A4 dimensions. */
 public Dimension = <A3>|<A4>;

values
 /** No values implemented. */


instance variables
 /** The title of the document. */
 private title: String := "Untitled document";

 /** The page count of the document. */
 private page_no: nat := 1;


 /** The selected color option for printing. */
 private color: Color := <COLOR>;

 /** The selected dimension size. */
 private dimension: Dimension := <A4>;


 /** A document always has one or more pages. It is impossible to have 0 or negative pages. */
 inv page_no >= 1;

operations
 /** The document constructor. */

 public Document: String * nat1 * Color * Dimension ==> Document
 Document(t, pn, c, d) == (title := t; page_no := pn; color := c; dimension := d; return self)
 pre len t <> 0 and pn > 0
 post self.title = t and self.page_no = pn and self.color = c and self.dimension = d and RESULT =
      self;

 /** Getter method for title. */

 public pure get_title: () ==> String
 get_title() == return self.title
 pre true
 post RESULT = self.title;

 /** Getter method for number of pages. */
 public pure get_page_no: () ==> nat
 get_page_no() == return self.page_no
 pre true
 post RESULT = self.page_no;

 /** Getter method for color. */
 public pure get_color: () ==> Color
 get_color() == return self.color
 pre true
 post RESULT = self.color;

 /** Getter method for dimension. */
 public pure get_dimension: () ==> Dimension
 get_dimension() == return self.dimension
 pre true
 post RESULT = self.dimension;

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */
```

```
end Document
```

Function or operation	Line	Coverage	Calls
Document	19	100.0%	24
get_color	37	100.0%	8
get_dimension	43	100.0%	8
get_page_no	31	100.0%	20
get_title	25	100.0%	2
Document.vdmpp		100.0%	62

2 Employee

```
/**
 * A worker for the printing service company whose main purpose is fixing printer malfunctions.
 */
class Employee

types
 /** Type definition for string. */
 public String = seq of char;

values
 /** No values implemented. */

instance variables
 /** The name of the employee. */

 private name: String := "Unnamed employee";

operations
 /** Employee constructor. */
 public Employee: String ==> Employee
 Employee(n) == (name := n; return self)

 pre len n <> 0
 post self.name = n and RESULT = self;

 /** Getter method for name. */
 public pure get_name: () ==> String
 get_name() == return self.name

 pre true
 post RESULT = self.name;

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */

end Employee
```

Function or operation	Line	Coverage	Calls
Employee	15	100.0%	10
assign	21	100.0%	10
get_name	27	100.0%	2
Employee.vdmpp		100.0%	22

3 Malfunction

```
/**
 * A malfunction report with a specific printer.
 * Usually reported by a student operating with the machine.
 */
class Malfunction

types
 /**
  * We defined 4 different common issues with printers:
  * NO_PAPER: no paper, NO_CMY: at least one colored ink cartridge empty,
  * NO_BLK: black cartridge empty, UNKNOWN: other.
  */
 public Issue = <NO_PAPER>|<NO_CMY>|<NO_BLK>|<UNKNOWN>;

 /** Type definition for string. */


 public String = seq of char;

values
 /** No values implemented. */

instance variables
 /** The specific issue of this malfunction report. */
 private issue: Issue := <UNKNOWN>;

 /** The printer ID associated to this malfunction. */
 private printer_id: String := "";


 /** Whether this malfunction was already solved or not by an employee. */
 private solved: bool := false;

operations
 /** The malfunction constructor. */
 public Malfunction: Issue * String ==> Malfunction
 Malfunction(i, s) == (issue := i; printer_id := s; return self)
 post issue = i and printer_id = s and solved = false and RESULT = self;

 /**
  * Closes this malfunction by fixing the machine.
  * After the service has assigned an employee to a malfunction, the employee fixes the machine.
  */
 public fix: () ==> ()
 fix() == (solved := true)
 pre (solved = false)
 post solved = true;

 /** Getter method for issue. */

 public pure get_issue: () ==> Issue
```

```
 get_issue() == return (self.issue)
 post RESULT = self.issue;

 /** Getter method for printer_id. */

 public pure get_printer_id: () ==> String
 get_printer_id() == return (self.printer_id)
 post RESULT = self.printer_id;

 /** Getter method for solved. */
 public pure get_solved: () ==> bool
 get_solved() == return (self.solved)
 post RESULT = self.solved;

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */

end Malfunction
```

Function or operation	Line	Coverage	Calls
Malfunction	16	100.0%	5
fix	16	100.0%	4
get_issue	47	100.0%	1
get_printer_id	52	100.0%	1
get_solved	28	100.0%	2
Malfunction.vdmpp		100.0%	13

4 Printer

```
/**
 * Represents a printer located in a university.
 */
class Printer

values
 /** The base price of printing a black & white page. */
 public BW_PAGE_PRICE: rat = 0.10;

 /** The base price of printing a colored page. */
 public COLOR_PAGE_PRICE: rat = 0.25;

 /** The multiplier applied to the above prices by printing in dimension A3. */
 public DIM_MULT_PRICE: rat = 2;

 /** A map relating university codes to the shortened version it should go on the ID. */
 public UNI_CODE: map University to String = {<FEUP>|->"fe", <ICBAS>|->"ic", <FLUP>|->"fl"};

 /** A utility map which converts digits to char. */
 public NAT_2_CHAR: map nat to char = {0|->'0', 1|->'1', 2|->'2', 3|->'3', 4|->'4', 5|->'5', 6|->
     '6', 7|->'7', 8|->'8', 9|->'9'}

types
```

```
 /** Type definition for string. */
 public String = seq of char;


 /** Currently, only the following 3 universities were made clients of this company. */
 public University = <FEUP>|<ICBAS>|<FLUP>;

 /** Statistics logged on this printer. */
 public Statistic = <STAT_DOCS_PRINTED>|<STAT_PAGES_PRINTED>;


values

 /** No values implemented. */

instance variables
 /**

  * A map which ensures every printer on each university has an unique ID.
  * Counts how many printers there are in each university.
  */
 private static uni_id: map University to nat := {<FEUP>|->0, <ICBAS>|->0, <FLUP>|->0};


 /**
  * The printer's ID. It is unique throughout every other printer provided by the service.
  * Follows the syntax <uni_code>{2}<num>{4} (e.g. fe0001).
  */
 private id: String := "Unnamed printer";

 /** The student authenticated in the machine whose documents may be printed. */
 private auth_student: [Student] := nil;


 /** The printer's queue. Documents in this queue may be printed. */
 private queue: seq of Document := [];

 /** A map which relates statistic labels to values. */
 private stats: map Statistic to nat := { |-> };

operations
 /** The printer constructor. */
 public Printer: University ==> Printer
 Printer(i) == (id := self.inc_uni_id(i); return self)
 post self.assert_id_syntax(id) and RESULT = self;

 /**
  * Increment university-scoped printer unique ID so that following printers have a incrementally

  * larger ID. It returns the new ID and increments the number of machines found in a university.
  */
 public inc_uni_id: University ==> String
 inc_uni_id(u) ==
  (dcl new_id: String := UNI_CODE(u) ^ "000" ^ [NAT_2_CHAR(uni_id(u))];
  uni_id := uni_id ++ {u |-> uni_id(u) + 1};
  return new_id;
 );

 /**
  * Authenticate student.
  * An abstraction of logging into the printer.
  */
 public auth: Student ==> ()
 auth(student) == (auth_student := student; queue := [])
 post self.auth_student <> nil and self.queue = [];
```

```
/**
 * Reset operation.

 * Unauthenticates the student and clears the machine's printing queue.
 */
public reset: () ==> ()
reset() == (auth_student := nil; queue := [])

post auth_student = nil and queue = [];

/**
 * Select student's documents from its personal printing queue.
 * Students may select their documents by providing the index on their personal printing queue.
 * Selected documents (given their ID is valid) are then inserted into the printer's printing
     queue.

 */
public sel_student_docs: seq of nat ==> ()
sel_student_docs(idx) == for i in idx do
 if i <= len self.auth_student.get_queue() and (i > 0)
 then queue := queue ^ [self.auth_student.get_queue()(i)]
pre len idx <> 0 and (self.auth_student <> nil)
post len self.queue = len queue~ + len idx;



/**
 * Print the documents in the printing queue.
 * Composed of multiple steps, printing the queue first checks whether the user has sufficient
     funds
 * to print it. If so, then the print cost is deducted from the authenticated student's account.
 * Statistics such as number of documents and pages printed are logged. Every document printed
     is
 * deleted from the student's printing queue and the printer's own queue is left empty.


 */
public print: () ==> ()
print() ==
 (if self.check_insufficient_funds() then return;
 auth_student.add_balance(-self.calc_print_cost());
 record_statistic(<STAT_DOCS_PRINTED>, len queue);
 for doc in self.queue do

  (record_statistic(<STAT_PAGES_PRINTED>, doc.get_page_no());
  auth_student.delete_document(doc);
  queue := tl queue;
 );
)
pre self.auth_student <> nil and (len queue > 0)
post true;

/**
 * Returns the sum of the printing cost of every document.
 * Naturally it takes into account whether the document is to be printed in black & white or
     color
 * and also its dimensions. These factors change the subtotal according on the base price
     defined
 * on the 'values' section above.
 * The result is rounded to two decimal places.
 */
public pure calc_print_cost: () ==> rat
calc_print_cost() ==
```

```
  (dcl op_cost: rat := 0.0;

  for doc in self.queue do
   (dcl subtotal: rat := 0.0;
   if doc.get_color() = <BW> then subtotal := subtotal + BW_PAGE_PRICE * doc.get_page_no()
   else subtotal := subtotal + COLOR_PAGE_PRICE * doc.get_page_no();
   if doc.get_dimension() = <A3> then subtotal := subtotal * DIM_MULT_PRICE;
   op_cost := op_cost + subtotal;
  );
  return floor (op_cost * 100) / 100;
 )
 post true;

 /**
  * Check whether the student has sufficient funds to print its desired documents.
  * This method returns true if so and false otherwise.
  */
 public check_insufficient_funds: () ==> bool
 check_insufficient_funds() == return calc_print_cost() > self.auth_student.get_balance()
 post true;

 /**
  * Record a given statistic.
  * Given a statistic tag and a value, said value is incremented on the statistic map.
  */
 public record_statistic: Statistic * nat ==> ()
 record_statistic(s, v) ==
  if s not in set dom stats then stats := stats ++ {s |-> v}
  else stats := stats ++ {s |-> stats(s) + v}
 post assert_statistics(s, v, stats, stats~);

 /** Getter method for ID. */
 public pure get_id: () ==> String
 get_id() == return (self.id)
 post RESULT = self.id;

 /** Getter method for authenticated student. */
 public pure get_auth_student: () ==> [Student]
 get_auth_student() == return (self.auth_student)
 post RESULT = self.auth_student;

 /** Getter method for queue. */
 public pure get_queue: () ==> seq of Document
 get_queue() == return (self.queue)
 post RESULT = self.queue;

 /** Getter method for a specific statistic. */
 public pure get_statistic: Statistic ==> nat
 get_statistic(s) == return self.stats(s)
 post RESULT = self.stats(s);

functions
 /**
  * Assert the full syntax of the printer's ID.
  * Follows the syntax <uni_code>{2}<num>{4} (e.g. fe0001).
  */
 public assert_id_syntax: String -> bool
 assert_id_syntax(id) == (
  (has_substring("fe", id) or has_substring("fl", id) or has_substring("ic", id)) and (len id =
      6)
 );

 /** Auxiliary method which returns true if string s1 is contained in string s2 and false
     otherwise. */
 public has_substring: String * String -> bool
```

```
 has_substring(s1, s2) == elems s1 inter elems s2 = elems s1;

 /**
  * Assert the logging of statistics.
  * Checks whether the new statistics value is equal to the old one plus the value attributed.
  */
 public assert_statistics: Statistic * nat * map Statistic to nat * map Statistic to nat -> bool
 assert_statistics(s, v, new_s, old_s) == (
  if s not in set dom old_s then (new_s(s) = v)
  else (new_s = old_s ++ {s |-> old_s(s) + v})
 );

traces
 /** No combinatorial test traces implemented. */

end Printer
```

Function or operation	Line	Coverage	Calls
Printer	26	100.0%	36
assert_id_syntax	108	100.0%	4
assert_statistics	108	100.0%	4
auth	32	100.0%	6
calc_print_cost	65	100.0%	8
check_insufficient_funds	82	100.0%	8
get_auth_student	88	100.0%	3
get_id	102	100.0%	11
get_queue	94	100.0%	6
get_statistic	102	100.0%	1
has_substring	115	100.0%	15
inc_uni_id	33	100.0%	12
print	50	100.0%	1
record_statistic	84	100.0%	3
reset	37	100.0%	2
sel_student_docs	42	100.0%	7
Printer.vdmpp		100.0%	127

5 PrintingQueue

```
class PrintingQueue

types
 /** No types implemented. */

values
 /** No values implemented. */

instance variables
 /** A sequence of documents inside this printing queue. */
 private documents: seq of Document := [];

operations
```

```
 /**
  * Add a document to the bottom of this printing queue.
  */
 public push_document: Document ==> ()
 push_document(document) == documents := documents ^ [document]
 pre true

 post document in set elems documents and len documents = len documents~ + 1;

 /**
  * Delete a document from this printing queue.
  */
 public delete_document: Document ==> ()
 delete_document(dead_doc) == (
  dcl new_queue: seq of Document := [], popped: bool := false;
  for this_doc in documents do
   if dead_doc <> this_doc or popped then new_queue := new_queue ^ [this_doc] else popped := true
       ;
  documents := new_queue;

 )
 pre dead_doc in set elems documents
 post len documents = len documents~ - 1;

 /** Getter method for documents. */
 public pure get_documents: () ==> seq of Document
 get_documents() == return self.documents
 pre true
 post RESULT = self.documents;

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */

end PrintingQueue
```

Function or operation	Line	Coverage	Calls
delete_document	20	100.0%	10
get_documents	31	100.0%	52
push_document	14	100.0%	10
PrintingQueue.vdmpp		100.0%	72

6 Service

```
/**
 * The printing service's root class.
 * Represents the company behind the distributed printing service.
 * This class should technically only be instantiated once during the execution of the software.
 */
class Service

types
 /** Currently, only the following 3 universities were made clients of this company. */
 public Uni = <FEUP>|<ICBAS>|<FLUP>;
```

```
values
 /** No values implemented. */


instance variables
 /** A set of employees working for the company whose sole purpose is fixing printer malfunctions
     . */
 private employees: set of Employee := {};

 /** A set of clients for the company (in our abstraction, they're called universities. */
 private universities: set of University := {};


 /** A map which assigns employees to printer malfunctions. */

 private assigned: map Employee to [Malfunction] := {|->};

operations
 /**
  * Adds an employee.

  * Performing this action not only pushes it to the company's employee set but also creates a
      new
  * key on the assigned map for the employee which points to nil.
  * This indicates said employee still hasn't been made in charge of fixing a malfunction just
      yet.
  */
 public add_employee: Employee ==> ()
 add_employee(e) == (employees := employees union {e}; assigned := assigned ++ {e |-> nil};)
 pre e not in set employees and e not in set dom assigned
 post e in set employees and e in set dom assigned;

 /**

  * Removes an employee.


  * Conversingly to adding an employee, this action removes given employee from the company's
      employee
  * set and unmaps prior assignments of the employee to a malfunction.
  */
 public remove_employee: Employee ==> ()
 remove_employee(e) == (employees := employees \ {e}; assigned := {e} <-: assigned;)

 pre e in set employees and e in set dom assigned
 post e not in set employees and e not in set dom assigned;

 /**
  * Assign a malfunction to an employee.
  * Note that the employee must already not be assigned to other malfunction.

  * In case a malfunction is pushed and there's no free employee, it is discarded.
  */
 public push_issue: Malfunction ==> ()
 push_issue(m) == (
  dcl logged: bool := false;
  for all emp in set dom assigned do
  if assigned(emp) = nil and not logged then (assigned := assigned ++ {emp |-> m}; logged := true
      );
 )
 pre m not in set rng assigned
 post true;

 /**
```

```
  * Fixes all malfunctions.
  * Essentially sends every employee to work on its assigned machine. At the end of this
      operation,
  * no employee should be assigned to any malfunction, meaning either no printer was
      malfunctioning
  * before or all already assigned malfunctions were fixed.
  */
 public fix_all_issues: () ==> ()
 fix_all_issues() == (
  for all emp in set dom assigned do
  if assigned(emp) <> nil then (assigned(emp).fix(); assigned := assigned ++ {emp |-> nil});
 )
 post rng assigned = {nil};

 /** Getter method for the employees set. */
 public pure get_employees: () ==> set of Employee
 get_employees() == return self.employees
 post RESULT = self.employees;

 /** Getter method for the assigned map. */
 public pure get_assigned: () ==> map Employee to [Malfunction]
 get_assigned() == return self.assigned
 post RESULT = self.assigned;

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */

end Service
```

Function or operation	Line	Coverage	Calls
Service	15	100.0%	2
add_employee	23	100.0%	4
fix_all_issues	39	100.0%	2
fix_issue	38	100.0%	1
get_assigned	39	100.0%	6
get_employees	50	100.0%	1
get_issue_queue	44	100.0%	4
push_issue	28	100.0%	4
remove_employee	22	100.0%	1
Service.vdmpp		100.0%	25

7 Student

```
/**
 * A user in the printing service.
 * Each user belongs to a university and may utilize its university printing services.
 */
class Student

types
 /** Type definition for string. */
```

```
 public String = seq of char;

values
 /** No values implemented. */

instance variables
 /**

  * The unique student ID follows the UP syntax, only differing on the two first characters,
      which are
  * reserved to the university code.
  */
 private id: String := "fe000000000";

 /** The student balance, used to print documents. */

 private balance: rat := 0.0;

 /** A personal printing queue. Students may add their documents to it and then send them to a
     printer. */
 private queue: PrintingQueue := new PrintingQueue();

 /**

  * A student may never be in debt towards the service.
  * If the student does not have enough funds to perform an operation in full, it won't be
      permitted until
  * they add funds to their accounts.
  */
 inv balance >= 0.0;


operations
 /** The student constructor. */
 public Student: String ==> Student
 Student(idd) == (id := idd; return self)
 pre assert_id_syntax(idd)
 post self.id = idd and RESULT = self;



 /**
  * Add or subtract funds from the student's account.
  * Used whenever a student adds funds to its account or funds are deducted from it after
      printing.
  */
 public add_balance: rat ==> ()

 add_balance(value) == balance := balance + value
 pre value <> 0
 post balance = balance~ + value;

 /**
  * Push a document to the printing queue.

  * Adds a provided document to the personal printing queue.
  */
 public push_document: Document ==> ()
 push_document(document) == queue.push_document(document)
 pre true
 post document in set elems queue.get_documents();

 /**
  * Delete a document from the printing queue.
  * Conversingly to push_document, deletes a provided document from the personal printing queue.
```

```
 */
public delete_document: Document ==> ()
delete_document(document) == queue.delete_document(document)
pre document in set elems queue.get_documents()
post true;

/**
 * Report a printer malfunction.
 * Whenever a student witnesses a printer malfunction, it may choose to report it to the service
 * which will dispatch an employee to fix it.
 * It must provide a Malfunction object as a report.
 */
public report_malfunction: Service * Malfunction ==> ()
report_malfunction(s, m) == s.push_issue(m)
post true;

/** Getter method for name. */
public pure get_id: () ==> String
get_id() == return self.id
pre true
post RESULT = self.id;

/** Getter method for balance. */
public pure get_balance: () ==> rat
get_balance() == return self.balance
pre true
post RESULT = self.balance;

/** Getter method for printing queue's documents. */
public pure get_queue: () ==> seq of Document
get_queue() == return self.queue.get_documents()
pre true
post RESULT = self.queue.get_documents();

functions

/**
 * Asserts whether provided ID follows the required syntax.
 * The unique student ID follows the UP syntax, only differing on the two first characters,
     which are
 * reserved to the university code.
 * Syntax: <uc>{2}<num>{9}
 */
public assert_id_syntax: String -> bool
assert_id_syntax(id) == ((len id = 11) and

 (has_substring("fe", id) or has_substring("fl", id) or has_substring("ic", id))
);

/** Auxiliary method which returns true if string s1 is contained in string s2 and false
    otherwise. */
public has_substring: String * String -> bool
has_substring(s1, s2) == elems s1 inter elems s2 = elems s1;

traces
/** No combinatorial test traces implemented. */

end Student
```

Function or operation	Line	Coverage	Calls

Student	16	100.0%	12
add_balance	22	100.0%	5
assert_id_syntax	97	100.0%	12
delete_document	34	100.0%	4
get_balance	52	100.0%	6
get_id	46	100.0%	8
get_queue	40	100.0%	19
has_substring	105	100.0%	22
push_document	28	100.0%	10
report_malfunction	41	100.0%	4
Student.vdmpp		100.0%	102

8 University

```
/**
 * An abstraction for the clients role in the printing service.
 */
class University

types
 /** Currently, only the following 3 universities were made clients of this company. */
 public Uni = <FEUP>|<ICBAS>|<FLUP>;

values
 /** No values implemented. */

instance variables
 /** This university's university code. */
 private name: Uni := <FEUP>;


 /** A set of students enrolled in this university who may use its printing service. */
 private students: set of Student := {};

 /** A set of printers available on this university. */
 private printers: set of Printer := {};


operations
 /** University constructor. */
 public University: Uni * set of Student * set of Printer ==> University
 University(n, s, p) == (name := n; students := s; printers := p; return self)
 pre s <> {} and p <> {}
 post self.name = n and self.students = s and self.printers = p and RESULT = self;

 /** Getter method for name. */
 public pure get_name: () ==> Uni
 get_name() == return self.name
 post RESULT = (self.name);

functions
 /** No functions implemented. */

traces
 /** No combinatorial test traces implemented. */

end University
```

Function or operation	Line	Coverage	Calls
University	16	100.0%	1
get_name	22	100.0%	2
University.vdmpp		100.0%	3

9 EmployeeTests

```
class EmployeeTests is subclass of TestSuite

types
-- TODO Define types here

values
-- TODO Define values here

instance variables
 employee1: Employee := new Employee("Miguel");
 employee2: Employee := new Employee("Ventura");

operations
 -- Tests whether names are assigned accordingly.

 private test_name_assignment: () ==> ()
 test_name_assignment() == (
  dcl university1: University := new University(<FEUP>, {new Student()}, {new Printer()});
  assert_true(university1.get_name() = <FEUP>);
  assert_true(employee1.get_name() = "Miguel");
  assert_true(employee2.get_name() = "Ventura");

 );

 public static main: () ==> ()
 main() == (
  dcl et: EmployeeTests := new EmployeeTests();
  et.test_name_assignment();
 )

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end EmployeeTests
```

Function or operation	Line	Coverage	Calls
main	21	100.0%	1
test_name_assignment	15	100.0%	1
EmployeeTests.vdmpp		100.0%	2

10 MalfunctionTests

```
class MalfunctionTests is subclass of TestSuite

types
-- TODO Define types here

values
-- TODO Define values here

instance variables
 service1: Service := new Service();
 employee1: Employee := new Employee("A");
 employee2: Employee := new Employee("B");
 employee3: Employee := new Employee("C");
 student1: Student := new Student("fe201503538");
 printer1: Printer := new Printer(<FEUP>);
 printer2: Printer := new Printer(<FEUP>);

operations


 private test_employee_addition: () ==> ()

 test_employee_addition() == (
  service1.add_employee(employee1);
  service1.add_employee(employee2);
  assert_true(service1.get_assigned() = {employee1 |-> nil, employee2 |-> nil});

  service1.add_employee(employee3);
  assert_true(service1.get_assigned() = {employee1 |-> nil, employee2 |-> nil, employee3 |-> nil
      });

  service1.remove_employee(employee2);
  assert_true(service1.get_assigned() = {employee1 |-> nil, employee3 |-> nil});
  assert_true(service1.get_employees() = {employee1, employee3});
 );

 -- Test #2: Tests whether reported malfunction is assigned to employee.
 private test_malfunction_assignment: () ==> ()
 test_malfunction_assignment() == (
  dcl malfunction1: Malfunction := new Malfunction(<NO_PAPER>, printer1.get_id());
  dcl malfunction2: Malfunction := new Malfunction(<NO_BLK>, printer1.get_id());
  dcl malfunction3: Malfunction := new Malfunction(<NO_CMY>, printer2.get_id());
  student1.report_malfunction(service1, malfunction1);
  assert_true(service1.get_assigned() = {employee1 |-> malfunction1, employee3 |-> nil});

  service1.fix_all_issues();
  assert_true(service1.get_assigned() = {employee1 |-> nil, employee3 |-> nil});

  student1.report_malfunction(service1, malfunction2);
  service1.add_employee(employee2);
  student1.report_malfunction(service1, malfunction3);
  assert_true(service1.get_assigned() = {employee1 |-> malfunction2, employee3 |-> nil, employee2
       |-> malfunction3});
 );

 -- Test #3: Tests whether fixing an issue closes the malfunction as solved.

 private test_malfunction_solved: () ==> ()
 test_malfunction_solved() == (
  dcl malfunction4: Malfunction := new Malfunction(<NO_PAPER>, printer1.get_id());
  student1.report_malfunction(service1, malfunction4);
  assert_true(malfunction4.get_solved() = false);

  service1.fix_all_issues();
```

```
 assert_true(malfunction4.get_solved() = true);
);

-- Test #4: Test malfunction variable extraction.

private test_malfunction_getters: () ==> ()
test_malfunction_getters() == (
 dcl malfunction5: Malfunction := new Malfunction(<NO_BLK>, printer1.get_id());
 assert_true(malfunction5.get_issue() = <NO_BLK>);
 assert_true(malfunction5.get_printer_id() = printer1.get_id());
);

public static main: () ==> ()
main() == (
 dcl mt: MalfunctionTests := new MalfunctionTests();
 mt.test_employee_addition();
 mt.test_malfunction_assignment();
 mt.test_malfunction_solved();
 mt.test_malfunction_getters();
);

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end MalfunctionTests
```

Function or operation	Line	Coverage	Calls
main	20	100.0%	1
test_employee_addition	19	100.0%	1
test_malfunction_assignment	19	100.0%	1
test_malfunction_getters	63	100.0%	1
test_malfunction_solved	52	100.0%	1
MalfunctionTests.vdmpp		100.0%	5

11 PrinterTests

```
class PrinterTests is subclass of TestSuite

types
-- TODO Define types here

values
-- TODO Define values here

instance variables
 private student1: Student := new Student("fe201503538");
 private student2: Student := new Student("ic201793252");

 document1: Document := new Document("d1", 20, <BW>, <A4>);
 document2: Document := new Document("d2", 40, <COLOR>, <A4>);
 document3: Document := new Document("d3", 60, <COLOR>, <A3>);
```

```
operations
 -- Test #1: Tests whether the printer's ID correctly increments relative to the university
     registered.
 private test_printer_id: () ==> ()
 test_printer_id() == (
  dcl printer1: Printer := new Printer(<FEUP>);
  dcl printer2: Printer := new Printer(<FEUP>);
  dcl printer3: Printer := new Printer(<ICBAS>);


  dcl printer4: Printer := new Printer(<FLUP>);
  dcl printer5: Printer := new Printer(<FEUP>);

  assert_true(printer1.get_id() = "fe0000");
  assert_true(printer2.get_id() = "fe0001");
  assert_true(printer3.get_id() = "ic0000");
  assert_true(printer4.get_id() = "fl0000");
  assert_true(printer5.get_id() = "fe0002");

  assert_true(document1.get_title() = "d1");
 );

 -- Test #2: Tests student authentication and unauthentication on the printer.

 private test_student_authentication: () ==> ()
 test_student_authentication() == (
  dcl printer1: Printer := new Printer(<FEUP>);
  printer1.auth(student1);
  assert_true(printer1.get_auth_student() = student1);

  printer1.auth(student2);
  assert_true(printer1.get_auth_student().get_id() = student2.get_id());

  printer1.reset();
  assert_true(printer1.get_auth_student() = nil);
 );


 -- Test #3: Tests whether documents selected on a student's PQ are added to printer's PQ.
 private test_queues_linking: () ==> ()
 test_queues_linking() == (
  dcl printer1: Printer := new Printer(<FEUP>);
  printer1.auth(student1);
  student1.push_document(document1);
  student1.push_document(document2);
  printer1.sel_student_docs([1]);
  assert_true(printer1.get_queue() = [document1]);


  student1.push_document(document3);
  printer1.sel_student_docs([3,1]);
  assert_true(printer1.get_queue() = [document1, document3, document1]);

  printer1.auth(student2);
  assert_true(printer1.get_queue() = []);

  student2.push_document(document1);
  printer1.sel_student_docs([1]);
  assert_true(printer1.get_queue() = [document1]);

  printer1.reset();
  assert_true(printer1.get_queue() = []);
 );

 -- Test #4: Tests document printing from the printer's printing queue.
```

```
 private test_printing: () ==> ()
 test_printing() == (
  dcl printer2: Printer := new Printer(<FEUP>);

  printer2.auth(student1);
  printer2.sel_student_docs([1]);
  assert_true(printer2.get_queue() = [document1] and student1.get_queue() = [document1, document2
      , document3]);

  printer2.auth(student2);
  student2.push_document(document1);
  student2.push_document(document3);
  printer2.sel_student_docs([1,3]);
  assert_true(printer2.calc_print_cost() = 32);

  student2.add_balance(10);
  printer2.print();
  assert_true(student2.get_balance() = 10);

  student2.add_balance(30);
  printer2.print();
  assert_true(printer2.get_statistic(<STAT_DOCS_PRINTED>) = 2);
  assert_true(student2.get_balance() + 32 = 40);
 );

 public static main: () ==> ()
 main() == (
  dcl pt: PrinterTests := new PrinterTests();
  pt.test_printer_id();
  pt.test_student_authentication();
  pt.test_queues_linking();
  pt.test_printing();
 );

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end PrinterTests
```

Function or operation	Line	Coverage	Calls
main	80	100.0%	1
test_printer_id	24	100.0%	1
test_printing	61	100.0%	1
test_queues_linking	37	100.0%	1
test_student_authentication	24	100.0%	1
PrinterTests.vdmpp		100.0%	5

12 StudentTests

```
class StudentTests is subclass of TestSuite

types
```

```
-- TODO Define types here

values
-- TODO Define values here

instance variables
 student1: Student := new Student("fe201503538");
 student2: Student := new Student("fl201407644");
 student3: Student := new Student("ic201793252");

 document1: Document := new Document("d1", 20, <BW>, <A4>);
 document2: Document := new Document("d2", 40, <COLOR>, <A3>);
 document3: Document := new Document("d3", 60, <COLOR>, <A4>);

operations
 -- Test #1: Tests adding or subtracting money from the student's balance.

 private test_balance_mod: () ==> ()
 test_balance_mod() == (
  student1.add_balance(4.25);
  assert_true(student1.get_balance() = 4.25);

  student1.add_balance(-3.25);
  assert_true(student1.get_balance() = 1.00);
 );

 -- Test #2: Tests adding and removing documents from the student's printing queue.
 private test_printing_queue: () ==> ()
 test_printing_queue() == (
  student1.push_document(document1);

  assert_true(student1.get_queue() = [document1]);

  student1.push_document(document1);
  student1.push_document(document2);
  assert_true(student1.get_queue() = [document1, document1, document2]);

  student1.delete_document(document1);
  assert_true(student1.get_queue() = [document1, document2]);

  student1.delete_document(document2);
  student1.push_document(document1);
  assert_true(student1.get_queue() = [document1, document1]);
 );

 -- Test #3: Tests correct retrieval of personal ID.
 private test_id_retrieval: () ==> ()
 test_id_retrieval() == (
  assert_true(student1.get_id() = "fe201503538");

  assert_true(student3.get_id() = "ic201793252");
 );

 public static main: () ==> ()
 main() == (
  dcl stt: StudentTests := new StudentTests();

  stt.test_balance_mod();
  stt.test_printing_queue();
  stt.test_id_retrieval();
 )

functions
-- TODO Define functiones here
```

```
traces
-- TODO Define Combinatorial Test Traces here

end StudentTests
```

Function or operation	Line	Coverage	Calls
main	57	100.0%	1
test_balance_mod	20	100.0%	1
test_id_retrieval	51	100.0%	1
test_printing_queue	33	100.0%	1
StudentTests.vdmpp		100.0%	4

13 TestSuite

```
class TestSuite

types
-- TODO Define types here

values
-- TODO Define values here

instance variables
-- TODO Define instance variables here

operations

 protected assert_true: bool ==> ()
 assert_true(b) == return
 pre b;

 -- Running the TestSuite runs every other test class that derives from it.

 public static main: () ==> ()
 main() == (
  new EmployeeTests().main();
  new StudentTests().main();
  new PrinterTests().main();
  new MalfunctionTests().main();
 );

functions
-- TODO Define functiones here

traces
-- TODO Define Combinatorial Test Traces here

end TestSuite
```

Function or operation	Line	Coverage	Calls
assert_true	13	100.0%	82

main	18	100.0%	2
TestSuite.vdmpp		100.0%	84