PerfectGym
Modelo Formal e Cobertura

Afonso Ramos
Cláudia Rodrigues

January 6, 2019

Contents

1 Exercise

```
class Exercise
types
 public ExerciseType = <Leg> | <Arm> | <Ab>;
instance variables
```

```
 protected load:nat;
 protected repetitions:nat1;
 protected type:ExerciseType;
 protected description:seq of char;

 inv len description> 0 and len description < 100;

operations

 -- Constructor

 public Exercise: nat * nat1 * ExerciseType * seq of char ==> Exercise
 Exercise(l, r, t, d) == (

  load := l;
  repetitions := r;
  type := t;
  description := d;
 )
 pre len d > 0 and len d < 100;

  -- Get Load

  public getLoad:() ==> nat
  getLoad() == return load;

  -- Get Repetitions

  public getRepetitions:() ==> nat
  getRepetitions() == return repetitions;

  -- Get Type

  public getType:() ==> ExerciseType
  getType() == return type;

  -- Get Description

  public getDescription:() ==> seq of char
  getDescription() == return description;

end Exercise
```

Function or operation	Line	Coverage	Calls
Exercise	16	100.0%	48
getDescription	39	100.0%	4
getLoad	27	100.0%	4
getRepetitions	31	100.0%	4
getType	35	100.0%	4
Exercise.vdmpp		100.0%	64

2 GymClass

```
class GymClass
```

```
values

 public classDuration: set of nat1 = {45, 60, 90}; --minutes
 public classCapacity: set of nat1 = {10, 20, 50}; --members

types

 -- Day in the week
 public Day_week = <Monday> | <Tuesday> | <Wednesday> | <Thursday> | <Friday> | <Saturday> | <
     Sunday>;

 -- Type of class
 public ClassType = <Cycling> | <BodyCombat> | <BodyAttack> | <Yoga> | <Zumba> | <RPM> | <Step>;

 -- Time in the day
 public Time:: hour : nat
        minute : nat

 inv t == t.hour < 24 and t.minute < 60;

 -- Duration
 public Duration = nat1
  inv d == d in set classDuration;

 -- Capacity
 public Capacity = nat1
  inv c == c in set classCapacity;

instance variables

 --Name
 private name: seq of char := [];
 private type: ClassType;

 -- Description
 private description: seq of char := [];

 -- Professor
 private professor:Professor;

 -- Participants
 private participants: set of Member := {};

 -- Date, time and duration
 private date: Day_week;
 private time: Time;
 private duration: Duration;

  -- Capacity
 private capacity: Capacity;

 -- available spots
 private availableSpace:nat;

 -- consistent available spots
 inv availableSpace = capacity - card participants - 1 and availableSpace >= 0;

  -- No empty name or description
  inv len name > 0 and len description > 0;

operations

  -- constructor
```

```
public GymClass : seq of char * ClassType* seq of char * Capacity * Professor * Day_week * Time
    * Duration ==> GymClass
GymClass (className, classType, classDescription, cap, prof, dt, tim, dur) == (
 name := className;
 type := classType;
 description := classDescription;
 capacity:= cap;
 professor := prof;
 date := dt;
 time := tim;
 duration := dur;
 availableSpace := capacity - 1; --professor
 return self
)
pre len className > 0 and len classDescription > 0;


-- get the class name

 pure public getName : () ==> seq of char
 getName () == (
  return name;
 );

 -- get the class type

 public getType : () ==> ClassType
 getType () == (
  return type;
 );

-- set the class name

 public setName :  seq of char ==> ()
 setName (n) == (
  name := n;
 );

 -- get the class description

 pure public getDescription : () ==>  seq of char
 getDescription () == (
  return description;
 );

 -- set the class description

public setDescription :  seq of char ==> ()
setDescription (d) == (
 description := d;
);

-- get all the participants

public pure getParticipants : () ==> set of Member
getParticipants() == (
 return participants;
);

-- get empty space

public getEmptySpace: () ==> nat
getEmptySpace()== (
 return availableSpace;
);
```

```
 -- add a new participant

 public addParticipant: Member ==> ()
 addParticipant(Member) == (

  atomic (
   participants := participants union {Member};
   availableSpace := availableSpace - 1;
   );
 )
 pre Member not in set participants and availableSpace > 0
 post Member in set participants and availableSpace = availableSpace~ -1 ;


 -- remove a participant

 public removeParticipant: Member ==> ()
 removeParticipant(Member) == (

  atomic (
   participants := participants \ {Member};
   availableSpace := availableSpace + 1;
  );
 )
 pre Member in set participants
 post participants = participants~ \ {Member}  and availableSpace = availableSpace~ +1 ;


 -- get date

 public pure getDate : () ==> Day_week
 getDate() == (
  return date;
 );

 -- get time

 public pure getTime : () ==> Time
 getTime() == (
  return time;
 );

 -- get duration

 public pure getDuration : () ==> nat1
 getDuration() == (
  return duration;
 );

 -- get capacity

 public pure getCapacity : () ==> nat1
 getCapacity() == (
  return capacity;
 );

 --get professor

 public pure getProfessor:() ==> Professor
 getProfessor() == (
  return professor;
 );

end GymClass
```

Function or operation	Line	Coverage	Calls
GymClass	65	100.0%	64
addParticipant	124	100.0%	16
getCapacity	168	100.0%	8
getDate	150	100.0%	280
getDescription	100	100.0%	8
getDuration	162	100.0%	44
getEmptySpace	118	100.0%	12
getName	82	100.0%	149
getParticipants	112	100.0%	88
getProfessor	174	100.0%	64
getTime	156	100.0%	404
getType	88	100.0%	96
removeParticipant	137	100.0%	8
setDescription	106	100.0%	4
setName	94	100.0%	4
GymClass.vdmpp		100.0%	1249

3 Member

```
class Member is subclass of User
types
values

instance variables

 -- Member's train plan
 private trainingPlan : [Plan] := nil;

 -- Member's weight
 private weight: real;

 -- Member's height
 private height: real;

 inv weight > 0 and height > 0;

 --User's referral
 private referral: int;

 -- Member's birth year
 private birthYear: nat1;


operations

 -- Constructor

 public Member : seq of char * seq of char * seq of char * Gender * nat1 * real * real * seq of
     char ==> Member
 Member (fName, lName, mail, g, year, w, h, pass) == (
  weight:= w;
  height:= h;
  birthYear:= year;
  referral := 0;
  User(fName, lName, mail, g, pass);
 )
 pre w >0 and h > 0;

 -- Set member's training plan

 public addTrainingPlan: Plan ==> ()
 addTrainingPlan(plan) == (
  trainingPlan := plan;
 );

 -- Get member's training plan

 public pure getTrainingPlan: () ==> [Plan]
 getTrainingPlan() == return trainingPlan;

 -- Get member weight

 public getWeight: () ==> real
 getWeight() == return weight;

 -- Get member height

 public getHeight: () ==> real
 getHeight() == return height;

 -- Set member weight

 public setWeight:real ==> ()
 setWeight(w) ==  weight:=w
  pre w > 0
  post weight=w;

 -- Set member height

 public setHeight: real ==> ()
 setHeight(h) == height:= h
  pre h > 0
  post height = h;

 -- Returns the user's referrals

 public getReferrals: () ==> int
 getReferrals() ==
  return referral;

 -- Increase user's referrals

 public setReferrals: () ==> ()
 setReferrals() == (
  referral := referral + 1;
 );

 -- Get member's monthly due

 public getMonthly: () ==> nat
 getMonthly() == (
  dcl age:nat := 2018 - birthYear;
  dcl ageDiscount:nat := 0;
  dcl monthly:nat;
```

```
  if age > 60 or age < 20
   then (ageDiscount := 1);
  monthly := floor (30 - 30 * getReferrals() / 30 - ageDiscount * 2/10 * 30);
  return monthly;
 );

end Member
```

Function or operation	Line	Coverage	Calls
Member	28	100.0%	64
addTrainingPlan	40	100.0%	4
getHeight	54	100.0%	24
getMonthly	81	100.0%	8
getReferrals	70	100.0%	12
getTrainingPlan	46	100.0%	8
getWeight	50	100.0%	24
setHeight	64	100.0%	8
setReferrals	75	100.0%	8
setWeight	58	100.0%	8
Member.vdmpp		100.0%	168

4 PerfectGym

```
class PerfectGym
instance variables

 -- Users
 users: map nat1 to User;
 loggedinUser : [User];

 -- Gym classes
 classes: set of GymClass;

 -- Invariants
 -- No two users with the same ID
 inv not exists u1, u2 in set rng users & u1 <> u2 and u1.getNumber() = u2.getNumber();

 -- Consistent map
 inv forall number in set dom users & users(number).getNumber()=number;

 -- No two gym classes with the same name
 inv not exists c1, c2 in set classes & c1 <> c1 and c1.getName() = c2.getName();

 -- Logged user belongs to users
 inv loggedinUser<>nil => loggedinUser in set rng users;

operations

 -- Constructor

 public PerfectGym : () ==> PerfectGym
 PerfectGym () == (
  users := {|->};
  classes := {};
  loggedinUser := nil;
 );

 -- Login member

  public loginMember: nat1 * seq of char ==> bool
  loginMember(membershipnumber, pass) == (

   if(userRegistered(membershipnumber)) then (

    dcl user:User := users(membershipnumber);

    if( user.getPassword() = pass ) then (
     loggedinUser := user;
     return true;
    )

   );

   return false;
  )
  pre len pass > 0 and len pass < 20 and loggedinUser = nil          -- only one user at a time
  post ( RESULT = true and loggedinUser <> nil) or RESULT = false;

  -- Log out member

  public logoutMember: () ==> ()
  logoutMember() == loggedinUser := nil
  pre loggedinUser <> nil
  post loggedinUser = nil;


  -- Get loggedinUser

  public pure getLoggedUser: () ==> [User]
  getLoggedUser() == (
    return loggedinUser;
  );

  -- Get users

  public getUsers: () ==> set of User
  getUsers() == (
    return rng users;
  );

  -- Checks if there is a user with a given membership number

  public pure userRegistered: nat1 ==> bool
  userRegistered(number) == (
    return number in set dom users;
  );

  --Checks if a user exists

  public pure userExists: User ==> bool
  userExists(user) == (
    return user in set rng users;
  );

  -- Get user according to membership number

  public getUser: nat1 ==> User
  getUser(number) == (
   return users(number);
  )
  pre userRegistered(number);

  -- Add a user if there is no user with the same membership number

  public addUser : User ==> bool
  addUser(u) == (

   if( not userRegistered(u.getNumber())) then (

     --add user
     users := users munion { u.getNumber() |-> u };

    return true;

   );
   return false;
  )
  post ( RESULT = true and users = users~ munion { u.getNumber() |-> u } ) or ( RESULT = false
      and users = users~ );

  -- Add a user if there is no user with the same membership number and has referral

  public addUserReferral : Member * User ==> bool
  addUserReferral(r,u) == (

   if( not userRegistered(u.getNumber())) then (

     --add user
     users := users munion { u.getNumber() |-> u };
     r.setReferrals();

    return true;

   );
   return false;
  )
 pre userExists(r)
  post ( RESULT = true and users = users~ munion { u.getNumber() |-> u } ) or ( RESULT = false
      and users = users~ );


 -- Get classes

 public getClasses: () ==> set of GymClass
 getClasses()==(
  return classes;
 );

 -- Get gym class from name

  public getGymClass: seq of char ==> [GymClass]
  getGymClass(name) == (

  for all gymclass in set classes do(
   if( gymclass.getName() = name) then
    return gymclass;
  );
  return nil;
  );

 -- Checks if there is a class with the same name

 public pure classRegistered: GymClass ==> bool
```

```
classRegistered(gclass) == (

 dcl name: seq of char := gclass.getName();
 for all gymclass in set classes do(

  if( gymclass.getName() = name) then (

   return true;

  );
 );
 return false;
);

-- Add a class if there is no class with the same name

public addClass: GymClass ==> ()
addClass (gclass) == (
 classes:= classes union {gclass};
)
pre gclass not in set classes
 and not classRegistered(gclass)
 and userExists(gclass.getProfessor())
 and Utilities`overlapClasses(gclass, classes)= false
post classes = classes~ union {gclass};


--Remove a class

public removeClass: GymClass ==> ()
removeClass(gclass) == (
 classes:= classes \ {gclass};
)
pre gclass in set classes
post classes = classes~ \ {gclass};


-- Get gym classes schedule

public getSchedule: () ==> map GymClass`Day_week to seq of GymClass
getSchedule() == (

 dcl result: map GymClass`Day_week to seq of GymClass := {|->};

 for all gclass in set classes do(

  dcl dayWeek: GymClass`Day_week := gclass.getDate();

  if(dayWeek not in set dom result) then (

   result:= result ++ {dayWeek|->[gclass]};

  )else(

   dcl list_aux:seq of GymClass := result(dayWeek);

   result:= result ++ {dayWeek|->list_aux ^ [gclass]};
  );
 );

 --order by time
 for all day in set dom result do(
  result(day):= Utilities`insertionSort(result(day));
 );
 return result;
);

-- Get gym classes in a given week day
public getSchedule: (GymClass`Day_week ) ==> map GymClass`Day_week to seq of GymClass
getSchedule(day) == (

  return {day} <: getSchedule();
);


-- Get gym schedule of a class type

public getSchedule2: (GymClass`ClassType) ==> map GymClass`Day_week to seq of GymClass`Time
getSchedule2(type) == (

 dcl tmp: map GymClass`Day_week to seq of GymClass := getSchedule();
 dcl result: map GymClass`Day_week to seq of GymClass`Time := {|->};

 --get times

 for all day in set dom tmp do(

   dcl gclasses:seq of GymClass := tmp(day);
   dcl times:seq of GymClass`Time:= [];

   dcl i:nat1:=1;
   while i < len gclasses + 1 do(

    dcl gclass:GymClass := gclasses(i);

    if(gclass.getType() = type) then(
     times := times ^ [gclass.getTime()];
    );

    i:= i +1;
   );

   if(len times > 0) then
   result:= result ++ {day|->times};
 );

 return result;
);

-- Get gym classes of a professor
public getClasses: Professor ==> set of GymClass
getClasses(prof) == (

 dcl result: set of GymClass:= {};

 for all gc in set classes do(

  if(prof = gc.getProfessor()) then result:= result union {gc};

 );
 return result;
);

 ----------------------------------------------------------------------
 ------------------ When the member is logged in ---------------------
 ----------------------------------------------------------------------

-- Enroll member in a gym class
```

```
public enrollGymClass: Member * GymClass ==> ()
enrollGymClass(member, gclass) == (
 gclass.addParticipant(member);
)
pre getLoggedUser() = member and gclass in set classes
post member in set gclass.getParticipants();


-- Remove member from a gym class

public removeUserGymClass: Member * GymClass ==> ()
removeUserGymClass(member, gclass) == (
 gclass.removeParticipant(member);
)
pre getLoggedUser() = member and gclass in set classes
post member not in set gclass.getParticipants();


-- Get gym classes of a member
public getClasses: Member ==> set of GymClass
getClasses(member) == (

 dcl result: set of GymClass:= {};

 for all gc in set classes do(

  if(member in set gc.getParticipants()) then result:= result union {gc};

 );
 return result;
)
pre getLoggedUser() = member;

-- Get training plan

public getPlan: Member ==> [Plan]
getPlan(member) == (

 return member.getTrainingPlan();
)
pre getLoggedUser() = member;


--Edit weight and height


public editWeight:Member * real ==> ()
editWeight(m, w) ==  (
 m.setWeight(w)
)
pre getLoggedUser() = m;


 public editHeight:Member * real ==> ()
editHeight(m, h) ==  (
 m.setHeight(h)
)
pre getLoggedUser() = m;



 ---------------------------------------------------------------------
 ------------------ When the professor is logged in ---------------------
 ---------------------------------------------------------------------
 -- Create a training plan for a member

 public createTrainingPlan: Professor * Member * Plan ==> ()
 createTrainingPlan(professor, member, plan) == (
   member.addTrainingPlan(plan);
 )
 pre getLoggedUser() = professor and userExists(member)
 post member.getTrainingPlan() = plan;

end PerfectGym
```

Function or operation	Line	Coverage	Calls
PerfectGym	27	100.0%	56
addClass	161	100.0%	42
addUser	93	100.0%	3
addUserReferral	109	100.0%	3
classRegistered	145	100.0%	57
createTrainingPlan	333	100.0%	9
editHeight	320	100.0%	6
editWeight	314	100.0%	3
enrollGymClass	271	100.0%	9
getClasses	128	100.0%	21
getGymClass	134	100.0%	5
getLoggedUser	62	100.0%	108
getPlan	304	100.0%	3
getSchedule	182	100.0%	3
getSchedule2	220	100.0%	15
getUser	86	100.0%	9
getUsers	68	100.0%	24
loginMember	35	100.0%	56
logoutMember	55	100.0%	8
removeClass	173	100.0%	3
removeUserGymClass	280	100.0%	3
userExists	80	100.0%	112
userRegistered	74	100.0%	119
PerfectGym.vdmpp		100.0%	677

5 Plan

```
class Plan

instance variables

 -- Exercises
 private series: seq of Exercise;

 -- Professor
 private professor:Professor;

operations
```

```
-- constructor

public Plan : seq of Exercise * Professor ==> Plan
Plan(ex, prof) == (
 series:= ex;
 professor:= prof;
)
pre len ex = card elems ex; --no same exercises


-- empty constructor
public Plan : () ==> Plan
Plan() == (
 series:= [];
);

-- get exercices

public getExercises:() ==> seq of Exercise
getExercises()  ==
 return series;

-- get professor

public getProfessor:() ==> Professor
getProfessor()  ==
 return professor;


-- add exercice to series

public addExercise: Exercise ==> ()
addExercise(ex) == (
  series:= series ^ [ex]
)
pre ex not in set elems series
post len series = len series~ +1 and series(len series) = ex;


-- remove exercise from series

public removeExercise:Exercise ==> ()
removeExercise(ex) == (

  dcl index:nat := 1;
  dcl exercises: seq of Exercise := series;
  dcl done:bool := false;

  while ( done = false ) do (
   if ( ex = hd exercises) then (

   done:=true;

   if( index = 1 ) then  series := tl series                 -- first element
   else if(index = len series) then series:= series(1,..., len series-1)   -- last element
   else series := series(1,..., index-1) ^ series(index+1, ..., len series)  -- middle element

  )  else(

   index := index +1;
   exercises := tl exercises;
  )
  )
)
```

```
 pre ex in set elems series
 post ex not in set elems series;

end Plan
```

Function or operation	Line	Coverage	Calls
Plan	14	100.0%	4
addExercise	40	100.0%	8
getExercises	29	100.0%	36
getProfessor	34	100.0%	16
removeExercise	49	100.0%	4
Plan.vdmpp		100.0%	68

6 Professor

```
class Professor is subclass of User
types

values

instance variables


operations

 -- Constructor

 public Professor : seq of char * seq of char * seq of char * Gender * seq of char ==> Professor
 Professor (fName, lName, mail, g, pass) == (

  User(fName, lName, mail, g, pass);

 )



end Professor
```

Function or operation	Line	Coverage	Calls
Professor	12	100.0%	52
Professor.vdmpp		100.0%	52

7 User

```
class User
types
 public Gender = <Masculine> | <Feminine>;
```

```
values

instance variables

 -- id (static)
 public static id : nat1 := 1;

 -- User's first name
 protected firstName: seq of char;

 -- User's last name
 protected lastName:seq of char;

 -- User's email
 protected email:seq of char;

 -- User's gender
 protected gender: Gender;

 --User's number
 protected membershipNumber : nat1;

 --User's mobile
 protected mobile: [nat1];

 --User's password
 protected password: seq of char;

operations

 -- Constructor

 public User : seq of char * seq of char * seq of char * Gender * seq of char  ==> User
 User (fName, lName, mail, g, pass) == (

  firstName := fName;
   lastName := lName;
   email := mail;
   gender := g;
   membershipNumber := id;
   id := id +1;
   mobile := nil;
   password := pass;
 )
 pre len mail >= 5 and len mail < 50
  and len fName > 0 and len fName < 20
  and len pass > 0 and len pass < 20
  and len lName > 0 and len lName < 20
 post firstName = fName and lastName = lName and password= pass and email = mail and gender = g
     and membershipNumber = id~;


 -- Returns the user's name

 public getName: () ==> seq of char
 getName() ==
  return firstName ^" "^ lastName;


 -- Returns the user's email

 public getEmail: () ==> seq of char
 getEmail() ==
  return email;
 -- Returns the user's membershipNumber

 public pure getNumber: () ==> nat1
 getNumber() ==
  return membershipNumber;


 -- Returns the user's gender

 public getGender: () ==> Gender
 getGender() ==
  return gender;


 -- Returns the user's password

 public getPassword: () ==> seq of char
 getPassword() == return password;


 -- Returns the user's mobile

 public getMobile: () ==> [nat1]
 getMobile() ==
  return mobile;


 -- Set mobile

 public setMobile: nat1 ==> ()
 setMobile(m) == (
  mobile:=m
 );

end User
```

Function or operation	Line	Coverage	Calls
User	35	100.0%	116
getEmail	61	100.0%	16
getGender	73	100.0%	16
getMobile	84	100.0%	8
getName	55	100.0%	16
getNumber	67	100.0%	4615
getPassword	79	100.0%	72
setMobile	90	100.0%	4
User.vdmpp		100.0%	4863

8 Utilities

```
class Utilities
operations

--Check is a gym class is before another
```

```
private static isBefore: GymClass * GymClass ==> bool
isBefore(gclass1, gclass2) == (

 if (gclass1.getTime().hour < gclass2.getTime().hour) then (
  return true
 )
 else if (gclass1.getTime().hour > gclass2.getTime().hour) then (
  return false
 )
 else ( --check minutes

  if(gclass1.getTime().minute < gclass2.getTime().minute)
     then return true else return false
 )
);

 -- Check if a class is in the same schedule

 public static pure overlapClasses: GymClass * set of GymClass ==> bool
 overlapClasses(gclass, classes) == (

  for all gymclass in set classes do(

   if( gymclass.getDate() = gclass.getDate()) then ( --same day of week

    let time1 = Utilities`timeToMinutes(gymclass.getTime().hour, gymclass.getTime().minute) ,
       time2 =  Utilities`timeToMinutes(gclass.getTime().hour, gclass.getTime().minute)  in (

     if( time1 < (time2 + gclass.getDuration()) and time2 < (time1 + gymclass.getDuration()) )
         then
       return true;
     )
   )
  );
  return false;
 );

--Sort gym classes by time

public static insertionSort: seq of GymClass ==> seq of GymClass
insertionSort(list) == (

 dcl i:nat:=1;
 dcl j:nat;
 dcl key:GymClass;
 dcl n:nat := len list;

 dcl result: seq of GymClass:= []; --ordered list
 result:= list;

 while ( i <= n ) do (

  key:= result(i);
  j:= i - 1;

  while( j>=1 and isBefore(key, result(j))) do(

   result(j+1):= result(j);
   j:= j - 1;

  );

   result(j+1):= key;
  i:= i +1;
 );
```

```
  return result;
);

functions
--Time in minutes

public timeToMinutes: nat * nat -> nat
timeToMinutes (hour, minute) == (
 (hour * 60) + minute
);

end Utilities
```

Function or operation	Line	Coverage	Calls
insertionSort	41	100.0%	132
isBefore	5	94.4%	84
overlapClasses	22	95.5%	28
timeToMinutes	73	100.0%	56
Utilities.vdmpp		97.0%	300

9 Main

```
class Main is subclass of MyTest
types
operations


 public static main: () ==> ()
 main() == (

    -- test user
   new TestUser().test();

   --test perfectgym
   new TestPerfectGym().test();

   --test gymclass
   new TestGymClass().test();

   --test exercise
   new TestExercise().test();

   --test training plan
   new TestPlan().test();
 );


end Main
```

Function or operation	Line	Coverage	Calls

main	5	100.0%	8
Main.vdmpp		100.0%	8

10 MyTest

```
class MyTest
operations

 -- Simulates assertion checking by reducing it to pre-condition checking.
 -- If 'arg' does not hold, a pre-condition violation will be signaled.

 protected assertTrue: bool ==> ()
 assertTrue(arg) ==
  return
 pre arg;

 -- Simulates assertion checking by reducing it to post-condition checking.
 -- If values are not equal, prints a message in the console and generates
 -- a post-conditions violation.

 protected assertEqual: ? * ? ==> ()
 assertEqual(expected, actual) ==
  if expected <> actual then (
      IO`print("Actual value (");
      IO`print(actual);
      IO`print(") different from expected (");
      IO`print(expected);
      IO`println(")\n")
  )
 post expected = actual

end MyTest
```

Function or operation	Line	Coverage	Calls
assertEqual	14	38.8%	0
assertTrue	6	100.0%	520
MyTest.vdmpp		45.0%	520

11 TestExercise

```
class TestExercise is subclass of MyTest
types
operations



  public newExercise: () ==> Exercise
 newExercise() == (
  return new Exercise(4, 6, <Leg>, "leg workout");
 );
 private createExercise: () ==> ()
 createExercise() == (

    dcl exercise:Exercise  := newExercise();
    assertEqual(exercise.getLoad(), 4);
    assertEqual(exercise.getRepetitions(), 6);
    assertEqual(exercise.getType(), <Leg>);
    assertEqual(exercise.getDescription(), "leg workout");
 );



 public test: () ==> ()
 test() == (

   createExercise();

 );


end TestExercise
```

Function or operation	Line	Coverage	Calls
createExercise	12	100.0%	4
newExercise	6	100.0%	4
test	23	100.0%	4
TestExercise.vdmpp		100.0%	12

12 TestGymClass

```
class TestGymClass is subclass of MyTest
types
operations


 public newGymClass: () ==> GymClass
 newGymClass() == (
   dcl user:Professor := new TestUser().newProfessor();
   return new GymClass("cycling", <Cycling>, "cycling class", 10, user, <Monday>, mk_GymClass`
       Time(15,20), 90);
 );

 -- Test create gym

 private createGymClass: () ==> ()
 createGymClass() == (

    dcl gclass:GymClass:= newGymClass();

     -- get
    assertEqual(gclass.getName(), "cycling");
    assertEqual(gclass.getType(), <Cycling>);
    assertEqual(gclass.getDescription(), "cycling class");
    assertEqual(gclass.getDate(), <Monday>);
    assertEqual(gclass.getCapacity(), 10);
    assertEqual(gclass.getTime(), mk_GymClass`Time(15,20));
    assertEqual(gclass.getDuration(), 90);

    -- set
    gclass.setName("run");
    gclass.setDescription("running class");
    assertEqual(gclass.getName(), "run");
    assertEqual(gclass.getDescription(), "running class");
 );

 -- Test add and remove participants

 private addParticipants: () ==> ()
 addParticipants() == (

    dcl gclass:GymClass:= newGymClass();
     dcl user:Member := new TestUser().newMember();

     assertEqual(card gclass.getParticipants(), 0);     --no participants
     assertEqual(gclass.getParticipants(), {});

     assertEqual( gclass.getCapacity(), 10);            -- capacity for 10
     assertEqual( gclass.getEmptySpace(), 9);         -- 9 spots left

     --add a participant
    gclass.addParticipant(user);
    assertEqual(card gclass.getParticipants(), 1);   --one participant
     assertEqual(gclass.getParticipants(), {user});

     assertEqual( gclass.getEmptySpace(), 8);         -- 8 spots left

   --remove a participant
    gclass.removeParticipant(user);
    assertEqual(card gclass.getParticipants(), 0);   --no participants
     assertEqual(gclass.getParticipants(), {});

     assertEqual( gclass.getEmptySpace(), 9);         -- 9 spots left
 );

 -- Test empty class name/description

 private changeClassName: () ==> ()
 changeClassName() == (

    dcl gclass:GymClass:= newGymClass();

     -- get
    assertEqual(gclass.getName(), "cycling");
    assertEqual(gclass.getDescription(), "cycling class");

    -- set
    gclass.setName("");      -- breaks invariant
    gclass.setDescription(""); -- breaks invariant

 );

 -- Test add same participant

  private addSameParticipant: () ==> ()
 addSameParticipant() == (

    dcl gclass:GymClass:= newGymClass();
     dcl user:Member := new TestUser().newMember();

     --add a participant
    gclass.addParticipant(user);
    assertEqual(card gclass.getParticipants(), 1);
     assertEqual(gclass.getParticipants(), {user});

    gclass.addParticipant(user);    -- breaks pre-condition
 );

 -- Test remove nonexisting participant

  private removeNonExistingParticipant: () ==> ()
 removeNonExistingParticipant() == (

    dcl gclass:GymClass:= newGymClass();
     dcl user:Member := new TestUser().newMember();
    gclass.removeParticipant(user);   -- breaks pre-condition
 );


 -- Test add participant to a full class

  private addParticipantFullClass: () ==> ()
 addParticipantFullClass() == (

   dcl gclass:GymClass:= newGymClass();
    dcl i:nat :=0;

    assertEqual( gclass.getCapacity(), 10);          -- capacity for 10
    assertEqual( card gclass.getParticipants(), 0);   -- 0 members
    assertEqual( gclass.getEmptySpace(), 9);         -- 9 spots left

     -- add 9 participants
     while i<9 do(
       dcl user:Member := new TestUser().newMember();
       gclass.addParticipant(user);
       i:= i+1;
    );

    assertEqual( card gclass.getParticipants(), 9);
    assertEqual( gclass.getEmptySpace(), 0);

    gclass.addParticipant(new TestUser().newMember()); -- breaks pre-condition
 );


 -- Runs all the tests associated with a gym class

 public test: () ==> ()
 test() == (

   createGymClass();
    addParticipants();

    /***** TEST CASES WITH INVALID INPUTS ******/
    --removeNonExistingParticipant();
    --changeClassName();
    --addSameParticipant();
    --addParticipantFullClass();
 );


end TestGymClass
```

Function or operation	Line	Coverage	Calls
addParticipantFullClass	103	0.0%	0
addParticipants	34	100.0%	4
addSameParticipant	78	0.0%	0
changeClassName	62	0.0%	0
createGymClass	12	100.0%	4
newGymClass	5	100.0%	8
removeNonExistingParticipant	93	0.0%	0
test	128	100.0%	4
TestGymClass.vdmpp		58.5%	20

13 TestPerfectGym

```
class TestPerfectGym is subclass of MyTest

operations


  public newGym: () ==> PerfectGym
 newGym() == (
  return new PerfectGym();
 );

 -- Test new gym

 private createGym: () ==> ()
 createGym() == (
  dcl gym:PerfectGym := newGym();
  assertEqual(gym.getUsers(), {});
  assertEqual(gym.getClasses(), {});
 );

 /*** USE CASE SCENARIO R01 - create user ***/
 -- Test add user

 private addUser: () ==> ()
 addUser() == (

    dcl gym:PerfectGym := newGym();
    dcl user:Member := new TestUser().newMember();

    assertTrue( not gym.userExists(user));
    assertTrue( gym.addUser(user));
    assertTrue( gym.userExists(user));

    assertEqual( gym.getUsers(), {user});

    assertEqual( gym.getUser(user.getNumber()), user);
 );

 /*** USE CASE SCENARIO R03 - user can edit weight and height ***/
 -- Test edit user

 private editUser: () ==> ()
 editUser() == (

    dcl gym:PerfectGym := newGym();
    dcl user:Member := new TestUser().newMember();
    assertTrue( gym.addUser(user));
    assertTrue( gym.userExists(user));

    --login
    assertTrue( gym.loginMember(user.getNumber(), user.getPassword()) = true);
     assertEqual( gym.getLoggedUser(), user);

    assertTrue(user.getWeight() = 50);
    assertTrue(user.getHeight() = 1.67);

    gym.editWeight(user, 51);
    gym.editHeight(user, 1.68);

    assertTrue(user.getWeight() = 51);
    assertTrue(user.getHeight() = 1.68);

);

/*** USE CASE SCENARIO R10 - create user with referral
        USE CASE SCENARIO R12 - consult monthly membership fee ***/

-- Test add user with referral

private addUserReferral: () ==> ()
addUserReferral() == (
  dcl gym:PerfectGym := newGym();
  dcl user:Member := new TestUser().newMember();
  dcl user2:Member := new TestUser().newMember2();
  dcl user3:Member := new TestUser().newMember3();
  dcl user4:Member := new TestUser().newMember4();

  assertTrue( not gym.userExists(user));
  assertTrue( gym.addUser(user));
  assertTrue( gym.addUser(user2));
  assertTrue( not gym.addUserReferral(user, user2));
  assertTrue( gym.addUserReferral(user, user3));
  assertTrue( gym.userExists(user));
  assertTrue( gym.userExists(user2));
  assertEqual( user.getReferrals(), 1);
  assertTrue( gym.addUserReferral(user, user4));

  -- Discount of 1$ per referral
  assertEqual( user.getMonthly(), 28);
  -- Discount of 20% due to age
  assertEqual( user2.getMonthly(), 24);

  assertEqual( gym.getUsers(), {user, user2, user3, user4});
);

-- Test add repeated user

private addRepeatedUser: () ==> ()
addRepeatedUser() == (

    dcl gym:PerfectGym := newGym();
    dcl user:Member := new TestUser().newMember();

    --number of users is 0
    assertEqual( card gym.getUsers(), 0);

    assertTrue( not gym.userExists(user));
    assertTrue( gym.addUser(user));
    assertTrue( gym.userExists(user));
```

```
   --number of users is 1
   assertEqual( card gym.getUsers(), 1);

   -- not add the user again
   assertEqual( gym.addUser(user), false);
   assertEqual( card gym.getUsers(), 1);
);

/*** USE CASE SCENARIO R09 - add or remove gym classes **/
-- Test add/remove classes

private addGymClass: () ==> ()
addGymClass() == (

 dcl gym:PerfectGym := newGym();
 dcl user:Professor := new TestUser().newProfessor();
 dcl gclass:GymClass := new GymClass("cycling", <Cycling>, "cycling class", 10, user, <Monday>,
     mk_GymClass`Time(15,20), 90);
 assertTrue (gym.addUser(user));

 assertEqual(gym.getClasses(), {});

 gym.addClass(gclass); --add
 assertEqual(gym.getClasses(), {gclass});
 assertTrue(gym.classRegistered(gclass));
 assertEqual(gym.getGymClass("cycling"), gclass);
 assertEqual(gym.getGymClass("sitting"), nil);

 gym.removeClass(gclass); --remove
 assertEqual(gym.getClasses(), {});
);


-- Add class with same name
private addGymClassSameName: () ==> ()
addGymClassSameName() == (

 dcl gym:PerfectGym := newGym();
 dcl user:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass := new GymClass("cycling", <Cycling>, "cycling class", 10, user, <Monday>,
      mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass := new GymClass("cycling", <Cycling>, "fit class", 10, user, <Monday>,
     mk_GymClass`Time(08,20), 90);
 assertTrue (gym.addUser(user));

 assertEqual(gym.getClasses(), {});

 gym.addClass(gclass1); --add
 assertEqual(gym.getClasses(), {gclass1});

 gym.addClass(gclass2); --breaks pre-condition
);


-- Test gym schedule
private testGymClasses: () ==> ()
testGymClasses() == (

 dcl gym:PerfectGym := newGym();
 dcl user:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass:= new GymClass("cycling",<Cycling>, "cycling class", 10, user, <Monday>,
     mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass:= new GymClass("yoga", <Yoga>, "yoga class", 10, user, <Monday>,
     mk_GymClass`Time(08,20), 90);
 assertTrue (gym.addUser(user));

 assertEqual(gym.getClasses(), {});
 gym.addClass(gclass1);
 assertEqual(gym.getClasses(), {gclass1});
 gym.addClass(gclass2);
 assertEqual(gym.getClasses(), {gclass1, gclass2});

);


/*** USE CASE SCENARIO R08 - user can see gym classes
       USE CASE SCENARIO R11 - user can see gym classes filtered by class type and week day **/


-- Test gym schedule
private testGymSchedule: () ==> ()
testGymSchedule() == (

 dcl gym:PerfectGym := newGym();
 dcl user:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass:= new GymClass("cycling", <Cycling>, "cycling class", 10, user, <Monday>,
     mk_GymClass`Time(15,20), 90);
 dcl gclass5:GymClass:= new GymClass("bodyattack", <BodyAttack>, "bodyattack class", 10, user, <
     Tuesday>, mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass := new GymClass("yoga", <Yoga>, "yoga class", 10, user, <Monday>,
     mk_GymClass`Time(08,00), 45);
 dcl gclass3:GymClass:= new GymClass("zumba", <Zumba>, "zumba class", 10, user, <Monday>,
     mk_GymClass`Time(20,30), 90);
 dcl gclass4:GymClass := new GymClass("cycling2", <Cycling>, "cycling class", 10, user, <
     Saturday>, mk_GymClass`Time(09,40), 60);
 dcl gclass6:GymClass:= new GymClass("cycling3", <Cycling>, "cycling class", 10, user, <Monday>,
      mk_GymClass`Time(08,50), 45);
 dcl schedule: map GymClass`Day_week to seq of GymClass;

 assertTrue (gym.addUser(user));

 schedule:= gym.getSchedule();

 --empty schedule
 assertEqual(gym.getClasses(), {});
 assertEqual(schedule, {|->});

 --add classes
 gym.addClass(gclass1);
 gym.addClass(gclass2);
 gym.addClass(gclass3);
 gym.addClass(gclass4);
 gym.addClass(gclass5);

 schedule:= gym.getSchedule();

 --ordered schedule
 assertEqual(gym.getClasses(), {gclass1, gclass2, gclass3, gclass4, gclass5});

 assertEqual(gym.getSchedule(<Monday>), {<Monday>|->[gclass2, gclass1, gclass3]});
 assertEqual(schedule, {<Monday>|->[gclass2, gclass1, gclass3], <Saturday>|->[gclass4], <Tuesday
     >|->[gclass5]});

 -- class shedule
 assertEqual(gym.getSchedule2(<Cycling>), {<Monday>|->[ mk_GymClass`Time(15,20)], <Saturday>|->[
      mk_GymClass`Time(09,40)] });
 gym.addClass(gclass6);
 assertEqual(gym.getSchedule2(<Cycling>), {<Monday>|->[ mk_GymClass`Time(08,50), mk_GymClass`
     Time(15,20)], <Saturday>|->[ mk_GymClass`Time(09,40)] });
 assertEqual(gym.getSchedule2(<BodyAttack>), {<Tuesday>|->[ mk_GymClass`Time(15,20)]});
 assertEqual(gym.getSchedule2(<RPM>), {|->});

);



-- Test overlap classes
private gymClassesTimeOverlap: () ==> ()
gymClassesTimeOverlap() == (

 dcl gym:PerfectGym := newGym();
 dcl user:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass:= new GymClass("cycling",  <Cycling>, "cycling
     class", 10, user, <Monday>, mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass:= new GymClass("yoga", <Yoga>, "yoga
     class", 10, user, <Monday>, mk_GymClass`Time(16,49), 90);

 assertTrue (gym.addUser(user));

 assertEqual(gym.getClasses(), {});
 gym.addClass(gclass1);
 assertEqual(gym.getClasses(), {gclass1});
 gym.addClass(gclass2);  --breaks pre condition

);

 /*** USE CASE SCENARIO R02 - login and logout user ***/

 -- Test login
 private testLogin: () ==> ()
testLogin() == (

  dcl gym:PerfectGym := newGym();
  dcl user:Member := new TestUser().newMember();
  assertTrue( gym.addUser(user));
  assertTrue( gym.userExists(user));
  assertEqual( gym.getLoggedUser(), nil);
  assertTrue( gym.loginMember(user.getNumber(), user.getPassword()) = true);
  assertEqual( gym.getLoggedUser(), user);
   gym.logoutMember();
   assertEqual( gym.getLoggedUser(), nil);

);


-- Test failed login
 private testFailedLogin: () ==> ()
testFailedLogin() == (

   dcl gym:PerfectGym := newGym();
  dcl user:Member := new TestUser().newMember();

  assertEqual( gym.getLoggedUser(), nil);
  assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), false); --user not
      registered
  assertEqual( gym.getLoggedUser(), nil);

  assertTrue( gym.addUser(user));
  assertTrue( gym.userExists(user));

  assertEqual( gym.loginMember(user.getNumber(), "wrongPassword"), false);   --wrong combination
  assertEqual( gym.getLoggedUser(), nil);

);
```

```
/*** USE CASE SCENARIO R04 - user can enroll in a class
      USE CASE SCENARIO R03 - user can access enrolled classes ***/


-- Test member add classes
public testAddGymClasses: () ==> ()
testAddGymClasses() == (

  dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass := new GymClass("cycling", <Cycling>, "cycling class", 10, prof, <Monday>,
      mk_GymClass`Time(15,20), 90);
 dcl user:Member := new TestUser().newMember();

 assertTrue( gym.addUser(user));
 assertTrue( gym.addUser(prof));
 gym.addClass( gclass1);


 -- 1. login user
 assertEqual( gym.getLoggedUser(), nil);
 assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), true);
 assertEqual( gym.getLoggedUser(), user);

 -- 2. enroll in gym class
 gym.enrollGymClass( user, gclass1);

 assertEqual( gclass1.getParticipants(), {user});
 assertEqual( gym.getClasses(user), {gclass1});

);

/*** USE CASE SCENARIO R04 - user can cancel a class **/

-- Test member remove classes
public testRemoveGymClasses: () ==> ()
testRemoveGymClasses() == (

  dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass := new GymClass("cycling", <Cycling>, "cycling class", 10, prof, <Monday>,
      mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass := new GymClass("cycling2",<Cycling>, "cycling class", 10, prof, <Tuesday
     >, mk_GymClass`Time(07,20), 90);

 dcl user:Member := new TestUser().newMember();
 assertTrue( gym.addUser(user));
 assertTrue( gym.addUser(prof));
 gym.addClass( gclass1);
 gym.addClass( gclass2);


 -- 1. login user
 assertEqual( gym.getLoggedUser(), nil);
 assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), true);
 assertEqual( gym.getLoggedUser(), user);

 -- 2. enroll in gym classes
 gym.enrollGymClass(user, gclass1);
 assertEqual( gclass1.getParticipants(), {user});
 assertEqual( gym.getClasses(user), {gclass1});
 gym.enrollGymClass(user, gclass2);
 assertEqual( gclass2.getParticipants(), {user});
 assertEqual( gym.getClasses(user), {gclass2, gclass1});

  -- 3. remove user from a gym class
  gym.removeUserGymClass(user, gclass1);
  assertEqual( gclass1.getParticipants(), {});
  assertEqual( gym.getClasses(user), {gclass2});
  assertEqual( gclass2.getParticipants(), {user});

);

/** USE CASE SCENARIO R11 - user can see gym classes filtered by professor**/

-- Test professor classes
public testProfessorClasses: () ==> ()
testProfessorClasses() == (

  dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass := new GymClass("cycling", <Cycling>, "cycling class", 10, prof, <Monday>,
      mk_GymClass`Time(15,20), 90);
 dcl gclass2:GymClass := new GymClass("cycling2", <Cycling>, "cycling class", 10, prof, <Tuesday
     >, mk_GymClass`Time(07,20), 90);

 assertTrue( gym.addUser(prof));
 gym.addClass( gclass1);
 gym.addClass( gclass2);

 assertEqual(gym.getClasses(prof), {gclass1, gclass2});

);


-- Test fail enroll class
public testFailEnrollGymClass: () ==> ()
testFailEnrollGymClass() == (

 dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl gclass1:GymClass := new GymClass("cycling",<Cycling>, "cycling class", 10, prof, <Monday>,
     mk_GymClass`Time(15,20), 90);

 dcl user:Member := new TestUser().newMember();

 assertTrue( gym.addUser(user));
 assertTrue( gym.addUser(prof));

 -- 1. login user
 assertEqual( gym.getLoggedUser(), nil);
 assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), true);
 assertEqual( gym.getLoggedUser(), user);

 -- 2. enroll in gym classes
 gym.enrollGymClass(user, gclass1);    -- breaks precondition (class not in the system)

  gym.logoutMember();
  assertEqual( gym.getLoggedUser(), nil);

  gym.addClass( gclass1);
  gym.enrollGymClass(user, gclass1);    -- breaks precondition (user not logged in)

);



/*** USE CASE SCENARIO R05 - professor can create a training plan for a member
```

```
   USE CASE SCENARIO R03 - user can access training plan
      USE CASE SCENARIO R06 - create exercises
      USE CASE SCENARIO R07 - add or remove exercises from plan ***/


-- Test add training plan
public testAddTrainingPlan: () ==> ()
testAddTrainingPlan() == (

 dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl user:Member := new TestUser().newMember();

 dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");
 dcl exercise2:Exercise:= new Exercise(2, 15, <Arm>, "arm workout");
 dcl plan:Plan := new Plan([exercise1, exercise2], prof);

 assertTrue( gym.addUser(user));
 assertTrue( gym.addUser(prof));

 -- 1. login prof
 assertEqual( gym.getLoggedUser(), nil);
 assertEqual( gym.loginMember(prof.getNumber(), prof.getPassword()), true);
 assertEqual( gym.getLoggedUser(), prof);

 -- 2. add that training plan to the user
 gym.createTrainingPlan(prof, user, plan);

 -- 3. user check the training plan
  gym.logoutMember();
  assertEqual( gym.getLoggedUser(), nil);
  assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), true);
  assertEqual( gym.getLoggedUser(), user);

  assertEqual(gym.getPlan(user), plan);

);


-- Test add training plan
public testPermissions: () ==> ()
testPermissions() == (

 dcl gym:PerfectGym := newGym();
 dcl prof:Professor := new TestUser().newProfessor();
 dcl user:Member := new TestUser().newMember();
  dcl user2:Member := new TestUser().newMember();

 dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");
 dcl exercise2:Exercise:= new Exercise(2, 15, <Arm>, "arm workout");
 dcl plan:Plan := new Plan([exercise1, exercise2], prof);

 assertTrue( gym.addUser(user));
 assertTrue( gym.addUser(user2));
 assertTrue( gym.addUser(prof));

 -- user login
 assertEqual( gym.getLoggedUser(), nil);
  assertEqual( gym.loginMember(user.getNumber(), user.getPassword()), true);
  assertEqual( gym.getLoggedUser(), user);

 -- add that training plan to the user
 gym.createTrainingPlan(prof, user, plan); --breaks precondition

  assertEqual(gym.getPlan(user), nil);
```

```
    assertEqual(gym.getPlan(user2), nil); --breaks precondition (cant see other members plan)

  );


  public test: () ==> ()
  test() == (

     createGym();
    addUser();
    editUser();
    addUserReferral();
    addRepeatedUser();
    addGymClass();
    testGymClasses();
    testGymSchedule();
    testLogin();
    testFailedLogin();
    testAddGymClasses();
    testRemoveGymClasses();
     testAddTrainingPlan();
     testProfessorClasses();

    /***** TEST CASES WITH INVALID INPUTS ******/
    --addGymClassSameName();
    --gymClassesTimeOverlap();
    --testFailEnrollGymClass();
    --testPermissions();
  );


end TestPerfectGym
```

Function or operation	Line	Coverage	Calls
addGymClass	115	100.0%	3
addGymClassSameName	135	0.0%	0
addRepeatedUser	92	100.0%	3
addUser	20	100.0%	6
addUserReferral	65	100.0%	3
createGym	11	100.0%	4
editUser	37	100.0%	9
gymClassesTimeOverlap	223	0.0%	0
newGym	5	100.0%	56
test	462	100.0%	2
testAddGymClasses	281	100.0%	2
testAddTrainingPlan	400	100.0%	6
testFailEnrollGymClass	364	0.0%	0
testFailedLogin	258	100.0%	6
testGymClasses	153	100.0%	6
testGymSchedule	176	100.0%	2
testLogin	242	100.0%	2
testPermissions	433	0.0%	0
testProfessorClasses	347	100.0%	2
testRemoveGymClasses	309	100.0%	2

TestPerfectGym.vdmpp		79.9%	114

14 TestPlan

```
class TestPlan is subclass of MyTest
types
operations


 -- Test create plan

 private createEmptyPlan: () ==> ()
 createEmptyPlan() == (

    dcl plan:Plan := new Plan();
    assertEqual(plan.getExercises(), []);
 );


 -- Test create plan

 private createPlan: () ==> ()
 createPlan() == (

    dcl series:seq of Exercise := [];
    dcl plan:Plan;
    dcl prof:Professor := new TestUser().newProfessor();
    dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");
    dcl exercise2:Exercise:= new Exercise(2, 15, <Arm>, "arm workout");

    series:= series ^ [exercise1];
    series:= series ^ [exercise2];

    plan := new Plan(series, prof);

    assertEqual(plan.getExercises(), [exercise1, exercise2]);
    assertEqual(plan.getProfessor(), prof);
 );

 -- Test add same exercise to plan

 private addSameExercise: () ==> ()
 addSameExercise() == (

    dcl series:seq of Exercise := [];
    dcl plan:Plan;
    dcl prof:Professor := new TestUser().newProfessor();
    dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");

    series:= series ^ [exercise1, exercise1];

    plan := new Plan(series, prof);     --breaks precondition

    plan := new Plan([exercise1], prof);
    plan.addExercise(exercise1);      --breaks precondition
 );


  -- Test add exercises

 private addExercises: () ==> ()
```

```
addExercises() == (

   dcl plan:Plan;
   dcl prof:Professor := new TestUser().newProfessor();
   dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");
   dcl exercise2:Exercise:= new Exercise(2, 15, <Arm>, "arm workout");
    dcl exercise3:Exercise:= new Exercise(2, 5, <Ab>, "ab workout");

   plan := new Plan([exercise1], prof);
   assertEqual(plan.getExercises(), [exercise1]);
   assertEqual(plan.getProfessor(), prof);

   plan.addExercise(exercise2);

   assertEqual(plan.getExercises(), [exercise1, exercise2]);
   assertEqual(plan.getProfessor(), prof);

   plan.addExercise(exercise3);

   assertEqual(plan.getExercises(), [exercise1, exercise2, exercise3]);
   assertEqual(plan.getProfessor(), prof);

);


-- Test remove exercises

private removeExercises: () ==> ()
removeExercises() == (

   dcl plan:Plan;
   dcl prof:Professor := new TestUser().newProfessor();
   dcl exercise1:Exercise:= new Exercise(4, 10, <Leg>, "leg workout");
   dcl exercise2:Exercise:= new Exercise(2, 15, <Arm>, "arm workout");
    dcl exercise3:Exercise:= new Exercise(2, 5, <Ab>, "ab workout");
    dcl exercise4:Exercise:= new Exercise(3, 5, <Ab>, "ab crunch");

   plan := new Plan([exercise1, exercise3, exercise2, exercise4], prof);

   assertEqual(plan.getExercises(), [exercise1, exercise3, exercise2, exercise4]);

   plan.removeExercise(exercise1);

   --remove 1st element
   assertEqual(plan.getExercises(), [exercise3, exercise2, exercise4]);

   plan.removeExercise(exercise2);
   assertEqual(plan.getExercises(), [exercise3, exercise4]);

   plan.removeExercise(exercise4);

   assertEqual(plan.getExercises(), [exercise3]);
);


-- Runs all the tests associated with a gym class

public test: () ==> ()
test() == (

  createEmptyPlan();
  createPlan();
  addExercises();
  removeExercises();
```

```
    /***** TEST CASES WITH INVALID INPUTS ******/
   --addSameExercise();

 );


end TestPlan
```

Function or operation	Line	Coverage	Calls
addExercises	53	100.0%	4
addSameExercise	35	0.0%	0
createEmptyPlan	7	100.0%	12
createPlan	16	100.0%	4
removeExercises	80	100.0%	4
test	109	100.0%	4
TestPlan.vdmpp		86.0%	28

15 TestUser

```
class TestUser is subclass of MyTest
types
operations

 --Creates a new member

 public newMember: () ==> Member
 newMember() == (
  return new Member(" C l udia ", "Rodrigues", "up201508262@fe.up.pt", <Feminine>, 1997, 50, 1.67,
       "qwerty1234");
 );


 public newMember2: () ==> Member
 newMember2() == (
  return new Member("Afonso", "Ramos", "up201506239@fe.up.pt", <Masculine>, 1950, 75, 1.91, "
      qwerty1234");
 );


 public newMember3: () ==> Member
 newMember3() == (
  return new Member("Carlos", "Freitas", "carlos@fe.up.pt", <Masculine>, 1997, 60, 1.51, "
      qwerty1234");
 );


 public newMember4: () ==> Member
 newMember4() == (
  return new Member("Pedro", "Sousa", "pedro@fe.up.pt", <Masculine>, 1997, 70, 1.71, "qwerty1234"
      );
 );

 --Creates a new professor

 public newProfessor: () ==> Professor
```

```
newProfessor() == (
 return new Professor("Jose", "Luis", "test@test.com", <Masculine>, "qwerty1234");
);

-- Test 1

private memberTest: () ==> ()
memberTest() == (

  dcl user:Member := newMember();
  assertTrue(user.getName() = " C l udia  Rodrigues");
  assertTrue(user.getEmail() = "up201508262@fe.up.pt");
  assertTrue(user.getPassword() = "qwerty1234");
  assertTrue(user.getGender() = <Feminine>);
  assertTrue(user.getNumber() = 1);

  assertTrue(user.getWeight() = 50);
  assertTrue(user.getHeight() = 1.67);
);

-- Test 2

private memberTest2: () ==> ()
memberTest2() == (

  dcl user:Member := newMember2();
  assertTrue(user.getName() = "Afonso Ramos");
  assertTrue(user.getEmail() = "up201506239@fe.up.pt");
  assertTrue(user.getPassword() = "qwerty1234");
  assertTrue(user.getGender() = <Masculine>);
  assertTrue(user.getNumber() = 2);

  assertTrue(user.getWeight() = 75);
  assertTrue(user.getHeight() = 1.91);
);

-- Test 3

private professorTest: () ==> ()
professorTest() == (

  dcl user:Professor := newProfessor();
  assertTrue(user.getName() = "Jose Luis");
  assertTrue(user.getEmail() = "test@test.com");
  assertTrue(user.getPassword() = "qwerty1234");
  assertTrue(user.getGender() = <Masculine>);
  assertTrue(user.getNumber() = 3);

  -- mobile test
  assertEqual(user.getMobile(), nil);
  user.setMobile(911111111);
  assertEqual(user.getMobile(), 911111111);

);

-- Test 4

private memberEditTest: () ==> ()
memberEditTest() == (

  dcl user:Member := newMember3();
  assertTrue(user.getName() = "Carlos Freitas");
  assertTrue(user.getEmail() = "carlos@fe.up.pt");
  assertTrue(user.getPassword() = "qwerty1234");
  assertTrue(user.getGender() = <Masculine>);
```

```
    assertTrue(user.getNumber() = 4);

    assertTrue(user.getWeight() = 60);
    assertTrue(user.getHeight() = 1.51);

    user.setWeight(65);
    user.setHeight(1.52);

    assertTrue(user.getWeight() = 65);
    assertTrue(user.getHeight() = 1.52);
  );


  -- Runs all the tests associated with a user

  public test: () ==> ()
  test() == (

    memberTest();
    memberTest2();
    professorTest();
    memberEditTest();
  );


end TestUser
```

Function or operation	Line	Coverage	Calls
memberEditTest	81	100.0%	6
memberTest	33	100.0%	4
memberTest2	48	100.0%	2
newMember	6	100.0%	22
newMember2	11	100.0%	4
newMember3	16	100.0%	4
newMember4	21	100.0%	2
newProfessor	27	100.0%	26
professorTest	63	100.0%	2
test	103	100.0%	2
TestUser.vdmpp		100.0%	74