Iron Trainers

Relatório

4º ano do Mestrado Integrado de Engenharia Informática e Computação

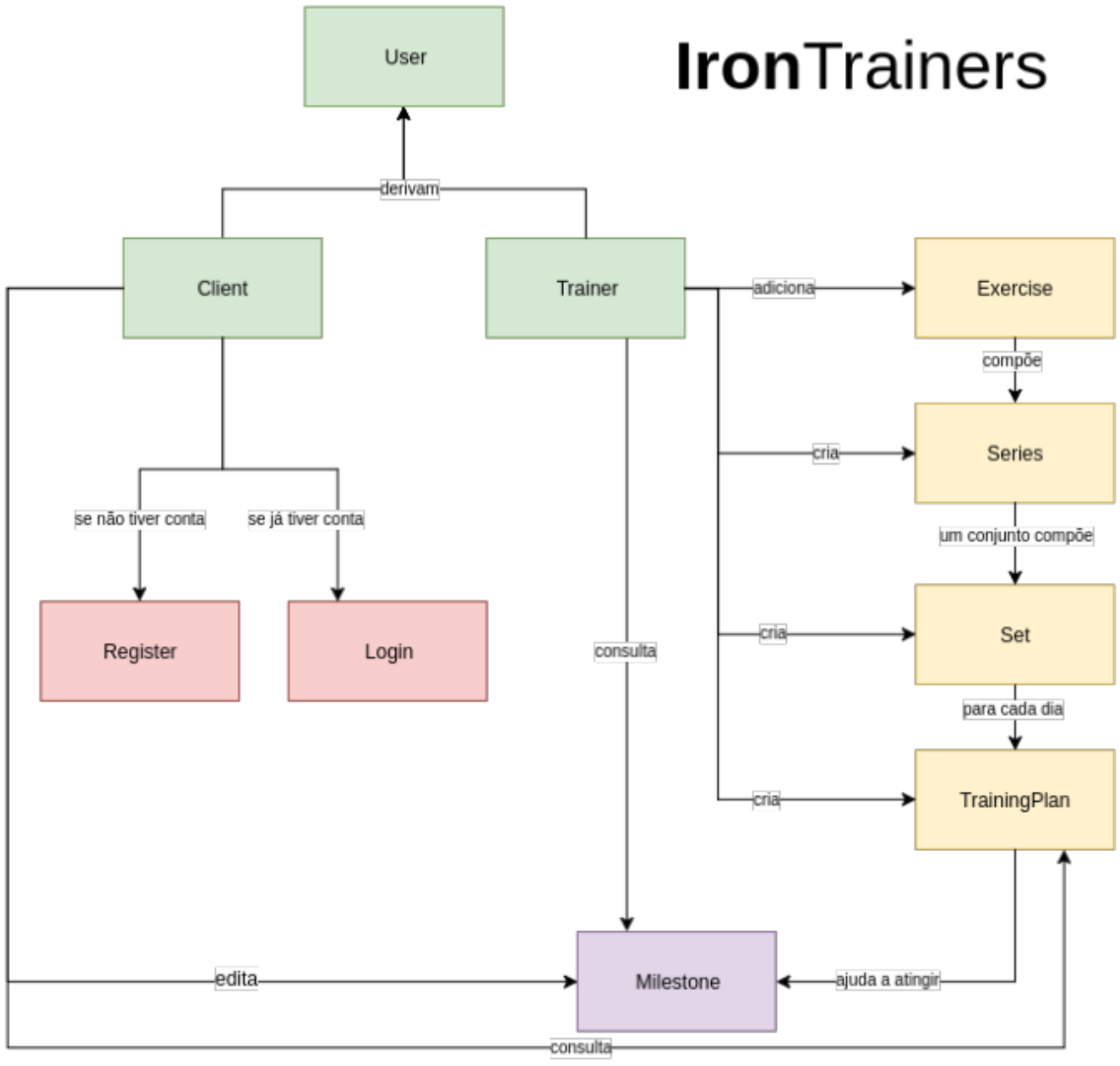
Métodos Formais de Engenharia de Software

Catarina Correia - up201405765 - up201405765@fe.up.pt
José Aleixo Cruz - up201403526 - up201403526@fe.up.pt

Janeiro 2018

Descrição informal do sistema e lista de requisitos

Descrição informal do sistema



Lista de requisitos

ID	**Prioridade**	**Descrição**
R1	Obrigatório	Permitir que uma pessoa se possa registar na plataforma como *Client*, utilizando um email e uma password. Um utilizador não consegue criar uma conta se o email já estiver registado no sistema.
R2	Obrigatório	Permitir que um utilizador registado (*Client* ou *Trainer*) possa fazer login e logout.

R3	Obrigatório	Permitir que um *Client* possa visualizar o seu próprio perfil.
R4	Obrigatório	Permitir que um *Client* possa editar as informações do seu perfil (peso e altura).
R5	Obrigatório	Permitir a um *Client* ver e/ou definir a sua *Milestone.*
R6	Obrigatório	Permitir a um *Client* consultar o *TrainingPlan* associado à sua *Milestone*.
R7	Obrigatório	Permitir a um Trainer visualizar todos os exercícios existentes no sistema.
R8	Obrigatório	Permitir a um *Trainer* adicionar um *Exercise*.
R9	Obrigatório	Permitir a um *Trainer* criar uma *Series*.
R10	Obrigatório	Permitir a um *Trainer* criar um *Set*.
R11	Obrigatório	Permitir a um *Trainer* criar um *TrainingPlan*.
R12	Obrigatório	Permitir a um *Trainer* procurar por um *Client.*
R13	Obrigatório	Permitir que um *Trainer* possa observar a *Milestone* e o *TrainingPlan* de um *Client.*
R14	Obrigatório	Permitir a um *Trainer* associar um *TrainingPlan* a uma *Milestone* de um *Client*.
R15	Opcional	Permitir ao utilizador converter unidades de massa de quilograma (kg) para libras (lbs).

Modelos UML

Modelo de casos de uso

Os casos de uso mais importantes são descritos em baixo.

Cenário	**Registo de um utilizador**
Descrição	Um utilizador não registado pode registar-se como *Client* para usufruir de todas as funcionalidades do sistema.
Pré-condições	1. O email inserido para registo não pode encontrar-se entre os utilizadores registados (*Client* e *Trainer*). 2. O email inserido tem entre 1 e 255 caracteres. 3. A password inserida tem entre 1 e 29 caracteres. 4. O nome tem entre 1 e 49 caracteres. 5. O peso e altura inseridos têm valores positivos. 6. A data de nascimento é menor que a data atual. 7. O género inserido só pode ser <F> (female) ou <M> (male).
Pós-condições	1. O email inserido passa a encontrar-se nos utilizadores registados. 2. O utilizador registado é um *Client.*
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Iniciar sessão**
Descrição	Um utilizador registado pode iniciar sessão para usufruir das funcionalidades do sistema. O login é diferente para utilizadores do tipo *Client* e *Trainer*.
Pré-condições	1. O email inserido para iniciar sessão encontra-se entre os utilizadores registados. 2. Não existe nenhum utilizador com sessão iniciada. 3. O email inserido tem entre 1 e 49 caracteres. Para iniciar sessão como *Client:* 1. O email inserido pertence à conta de um *Client*.

	Para iniciar sessão como *Trainer:* 2. O email inserido pertence à conta de um *Trainer*.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Terminar sessão**
Descrição	Um utilizador com sessão iniciada pode terminar sessão.
Pré-condições	1. Existe um utilizador com sessão iniciada.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Visualizar perfil**
Descrição	Um *Client* pode visualizar o seu próprio perfil com a sua informação pessoal.
Pré-condições	1. O utilizador deve ter sessão iniciada. 2. O utilizador deve ser um *Client.*
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Editar perfil**
Descrição	Um *Client* pode editar o seu próprio perfil (peso e/ou altura)
Pré-condições	1. O utilizador deve ter sessão iniciada. 2. O utilizador deve ser um *Client.*

	3. A altura (ou peso) inserida deve ter valores positivos.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Editar a *Milestone***
Descrição	Um *Client* pode editar a sua *Milestone* (peso que deseja atingir).
Pré-condições	1. O utilizador deve ter sessão iniciada. 2. O utilizador com sessão iniciada deve ser um *Client*. 3. O "peso que se deseja atingir " inserido deve ser positivo.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

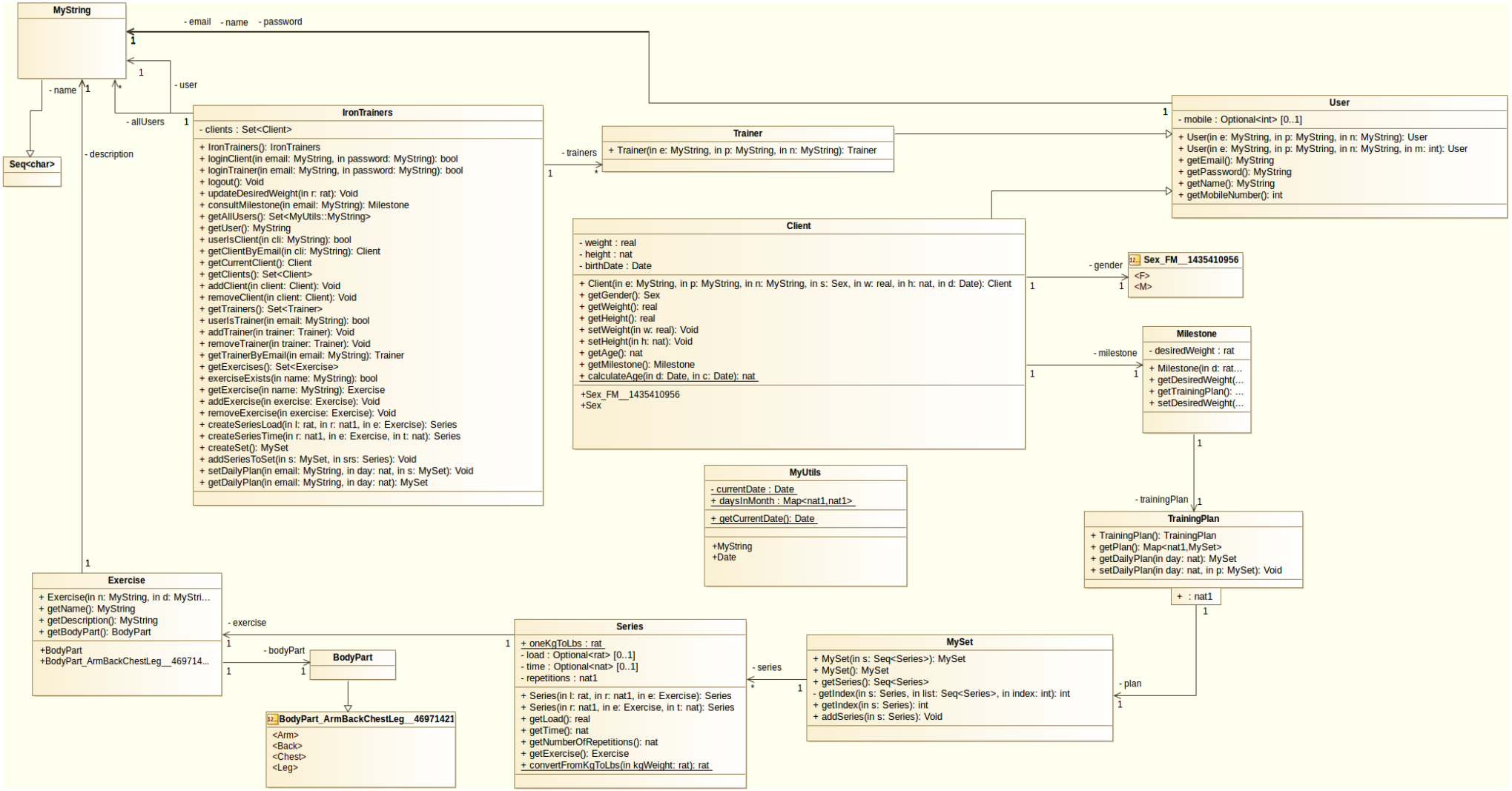
Cenário	**Consultar uma *Milestone***
Descrição	Um utilizador registado pode consultar uma *Milestone*.
Pré-condições	1. O utilizador deve ter sessão iniciada. Caso o utilizador seja um *Client*: 2. O utilizador com sessão iniciada deve ser um *Client*. Caso o utilizador seja um *Trainer*: 3. O utilizador com sessão iniciada deve ser um *Trainer*.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

Cenário	**Associar um *TrainingPlan* a uma *Milestone***
Descrição	Um treinador, com base no peso e peso que um *Client* pretende atingir, associa um plano de treino a esse mesmo cliente.
Pré-condições	1. O utilizador tem sessão iniciada. 2. O utilizador com sessão iniciada é um *Trainer*. 3. O utilizador cujo treino é suposto ser alterado deve ser um *Client*.
Pós-condições	(unspecified)
Passos	(unspecified)
Exceções	(unspecified)

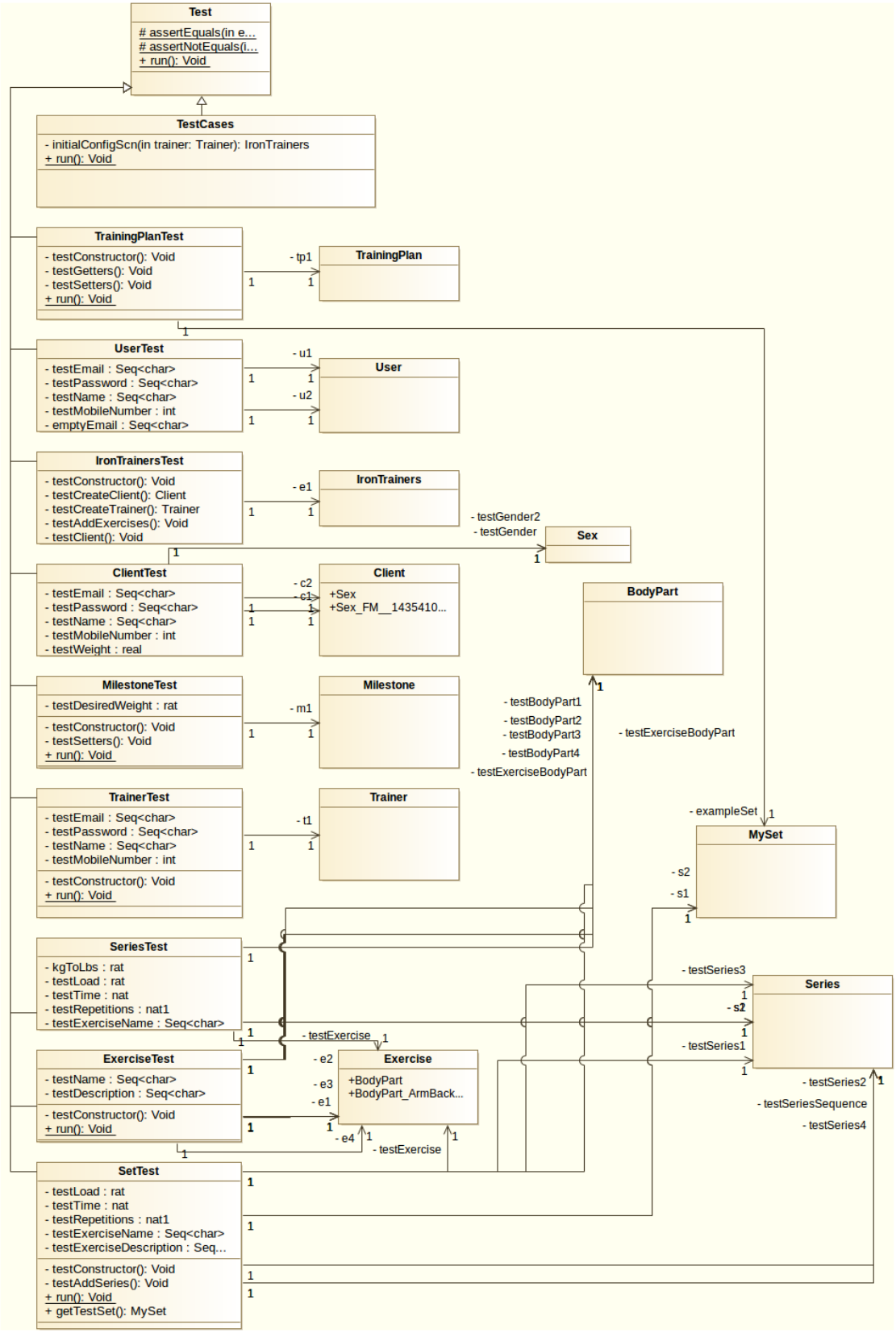
Cenário	**Adicionar um novo exercício**
Descrição	Um treinador adiciona um exercício ao sistema.
Pré-condições	1. O utilizador deve ter sessão iniciada. 2. O utilizador com sessão iniciada deve ser um *Trainer*. 3. O exercício que se pretende inserir não deve encontrar-se no sistema.
Pós-condições	1. O exercício inserido encontra-se na lista de exercícios existentes.
Passos	(unspecified)
Exceções	(unspecified)

Modelo de classes

Classes



Classes de teste



Modelo formal

User

```
class User
instance variables
	email : MyUtils`MyString;
	password : MyUtils`MyString;
	name : MyUtils`MyString;
	mobile : [int];
operations
	-- Constructor of user without mobile phone
	public User : MyUtils`MyString * MyUtils`MyString * MyUtils`MyString ==> User
	User(e, p, n) == (
		email := e;
		password := p;
		name := n;
		mobile := nil;
	)
	pre len e > 0 and len e < 256
		and len p > 0 and len p < 30
		and len n > 0 and len n < 50;

	-- Constructor of user with mobile phone
	public User : MyUtils`MyString * MyUtils`MyString * MyUtils`MyString * int ==> User
	User(e, p, n, m) == (
		email := e;
		password := p;
		name := n;
		mobile := m;
	)
	pre len e > 0 and len e < 256
		and len p > 0 and len p < 30
		and len n > 0 and len n < 50
		and (m div 10**8) = 9;	-- begins with 9 and has 9 numbers

	-- Get email of user
	pure public getEmail : () ==> MyUtils`MyString
	getEmail() == return email;

	-- Get password of user
	public getPassword : () ==> MyUtils`MyString
	getPassword() == return password;

	-- Get name of user
	public getName : () ==> MyUtils`MyString
	getName() == return name;

	public getMobileNumber : () ==> int
	getMobileNumber() == return mobile;
end User
```

Client

```
class Client is subclass of User
types
	public Sex = <F> | <M>
```

```
        inv s == s = <F> or s = <M>;
instance variables
    gender : Sex;
    weight :  real;
    height : nat;
    birthDate : MyUtils`Date;
    milestone : Milestone;

    -- Invariants
    inv weight > 0 and height > 0;
operations
    -- Constructor
    public Client : MyUtils`MyString * MyUtils`MyString * MyUtils`MyString * Sex * real * nat *
MyUtils`Date ==> Client
    Client(e, p, n, s, w, h, d) == (
        gender := s;
        weight := w;
        height := h;
        birthDate := d;
        milestone := new Milestone(0.0);
        User(e, p, n);
    )
    pre w > 0 and h > 0
    and d.day <= MyUtils`daysInMonth(d.month);

    -- Get gender
    public getGender : () ==> Sex
    getGender() == return gender
    pre not gender = nil;

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

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

    -- Set weight
    public setWeight : real ==> ()
    setWeight(w) == weight := w
    pre not w <= 0;

    -- Set height
    public setHeight : nat ==> ()
    setHeight(h) == height := h
    pre not h <= 0;

    -- Get age
    public getAge : () ==> nat
    getAge() == return calculateAge(birthDate, MyUtils`getCurrentDate());

    -- Get milestone
    public getMilestone : () ==> Milestone
    getMilestone() == return milestone;


functions
    public calculateAge : MyUtils`Date * MyUtils`Date -> nat
    calculateAge(d, c) ==
```

```
                c.year - d.year
        pre c.year > d.year and d.year > 0
                and c.month > 0 and c.month <= 12
                and d.month > 0 and d.month <= 12
                and c.day > 0 and c.day <= 31
                and d.day > 0 and d.day <= 31;
end Client
```

Trainer

```
class Trainer is subclass of User
operations
        public Trainer : MyUtils`MyString * MyUtils`MyString * MyUtils`MyString ==> Trainer
        Trainer(e, p, n) == (
                User(e, p, n);
        );
end Trainer
```

Exercise

```
class Exercise
types
        public BodyPart = <Leg> | <Arm> | <Chest> | <Back>
                inv b == b = <Leg> or b = <Arm> or b = <Chest> or b = <Back>;
instance variables
        name : MyUtils`MyString;
        description : MyUtils`MyString;
        bodyPart : BodyPart;

        inv len name > 0 and len name < 50
        and len description > 0 and len description < 250;

operations
        public Exercise : MyUtils`MyString * MyUtils`MyString * BodyPart ==> Exercise
        Exercise(n, d, b) == (
                name := n;
                description := d;
                bodyPart := b;
        )
        pre len n > 0 and len n < 50
                and len d > 0 and len d < 250;

        public pure getName : () ==> MyUtils`MyString
        getName() == return name;

        public getDescription : () ==> MyUtils`MyString
        getDescription() == return description;

        public getBodyPart : () ==> BodyPart
        getBodyPart() == return bodyPart;
end Exercise
```

Series

```
class Series
values
        public oneKgToLbs = 2.20462262;
```

```
instance variables
	load : [rat];
	time : [nat];
	repetitions: nat1;
	exercise : Exercise;

	inv repetitions > 0;
operations
	public Series : rat * nat1 * Exercise ==> Series
	Series(l, r, e) == (
		load := l;
		repetitions:= r;
		exercise := e;
		time := nil;
	)
	pre l > 0 and r > 0
	post load > 0;

	public Series : nat1 * Exercise * nat ==> Series
	Series(r, e, t) == (
		repetitions:= r;
		exercise := e;
		time := t;
		load := nil;
	)
	pre t > 0 and r > 0
	post time > 0;

	public getLoad : () ==> real
	getLoad() == return load
	pre load <> nil;

	public getTime : () ==> nat
	getTime() == return time
	pre time <> nil;

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

	public getExercise : () ==> Exercise
	getExercise() == return exercise;

functions
	public convertFromKgToLbs: rat -> rat
	convertFromKgToLbs(kgWeight) == kgWeight * oneKgToLbs
	pre kgWeight > 0;

end Series
```

MySet

```
class MySet

instance variables
	series : seq of Series;
operations
	public MySet : seq of Series ==> MySet
	MySet(s) == (
		series := s;
```

```
	);

	public MySet : () ==> MySet
	MySet() == (
		series := [];
	);

	public getSeries : () ==> seq of Series
	getSeries() == return series;

	-- Verifies if an element s exists in seq. If so, returns its index, else returns -1.
	private getIndex : Series * seq of Series * int ==> int
	getIndex(s, list, index) ==

		if len list = 0
		then return -1

		else if (s = hd list)
		then return index

		else getIndex(s, tl list, index + 1)
	pre index >= 0 and len list >= 0;

	-- Verifies if an element s exists in seq. If so, returns its index, else returns -1.
	public getIndex : Series ==> int
	getIndex(s) ==
		return getIndex(s, series, 0);

	-- Adds series to series
	public addSeries : Series ==> ()
	addSeries(s) ==
		if getIndex(s) = -1
		then series		:= series ^ [s];
end MySet
```

TrainingPlan

```
class TrainingPlan
instance variables
	plan : map nat1 to MySet;

operations
	public TrainingPlan : () ==> TrainingPlan
	TrainingPlan() == (
		plan := { |-> }; -- empty map
	);

	public getPlan : () ==> map nat1 to MySet
	getPlan() ==
		return plan;

	public getDailyPlan : nat ==> MySet
	getDailyPlan(day) ==
		return plan(day)
		pre day > 0;

	public setDailyPlan : nat * MySet ==> ()
	setDailyPlan(day, p) ==
		plan(day) := p
```

```
            pre day > 0;
end TrainingPlan
```

Milestone

```
class Milestone
instance variables
    desiredWeight : rat;
    trainingPlan : TrainingPlan;

    -- Invariants
    inv desiredWeight >= 0;

operations
    public Milestone : rat ==> Milestone
    Milestone(d) == (
        desiredWeight := d;
        trainingPlan := new TrainingPlan();
    )
    pre d >= 0;

    public getDesiredWeight : () ==> rat
    getDesiredWeight() == return desiredWeight;

    public getTrainingPlan : () ==> TrainingPlan
    getTrainingPlan() == return trainingPlan;

    public setDesiredWeight : rat ==> ()
    setDesiredWeight(w) ==
        desiredWeight := w
    pre w >= 0;

end Milestone
```

MyUtils

```
class MyUtils
types
    public MyString = seq1 of char;

    public Date::
        day : nat1
        month: nat1
        year : nat
        inv d == d.month > 0 and d.month <= 12
            and d.day > 0 and d.day <= 31
            and d.year > 1900;

values
    private currentDate = mk_Date(3, 1, 2018);
instance variables
    public static daysInMonth : map nat1 to nat1 := {
        1 |-> 31,
        2 |-> 29,
        3 |-> 31,
        4 |-> 30,
        5 |-> 31,
        6 |-> 30,
```

```
                7 |-> 31,
                8 |-> 31,
                9 |-> 30,
                10 |-> 31,
                11 |-> 30,
                12 |-> 31
        };
operations
        public static getCurrentDate: () ==> Date
        getCurrentDate() ==
        (
                return currentDate;
        );
end MyUtils
```

IronTrainers

```
class IronTrainers

instance variables
        clients : set of Client;
        trainers : set of Trainer;
        allUsers : set of MyUtils`MyString;
        exercises : set of Exercise;
        user : MyUtils`MyString;

operations

        public IronTrainers : () ==> IronTrainers
        IronTrainers() == (
                clients := {};
                trainers := {};
                allUsers := {};
                exercises := {};
                user := "undefined";
        );



        /*********************************************************/
        /******************        LOGIN & LOGOUT    ******************/
        /*********************************************************/


        -- Login in the application
        public loginClient : MyUtils`MyString * MyUtils`MyString ==> bool
        loginClient(email, password) ==
                if getClientByEmail(email).getPassword() = password
                then (
                        user := email;
                        return true;
                )
                else return false
        pre len email > 0 and len email < 50
                and {email} inter allUsers <> {}
```

```
        and userIsClient(email) and user = "undefined";

-- Login in the application
public loginTrainer : MyUtils`MyString * MyUtils`MyString ==> bool
loginTrainer(email, password) ==

        if getTrainerByEmail(email).getPassword() = password
        then (
                user := email;
                return true;
        )
        else return false
pre len email > 0 and len email < 50
        and {email} inter allUsers <> {}
        and userIsTrainer(email) and user = "undefined";

-- Logout from the application
public logout : () ==> ()
logout() ==
        user := "undefined"
pre not user = "undefined"
        and {user} inter allUsers <> {};


/*********************************************************/
/*********************  MILESTONE  *********************/
/*********************************************************/

-- A client updates its own desired weight
public updateDesiredWeight : rat ==> ()
updateDesiredWeight(r) ==
        let cli = getClientByEmail(user)
        in
                if isofclass(Client, cli)
                then cli.getMilestone().setDesiredWeight(r)
pre userIsClient(user) = true and user <> "undefined";

-- An user consults its own milestone
public consultMilestone : MyUtils`MyString ==> Milestone
consultMilestone(email) ==
        let cli = getClientByEmail(email)
        in
                if isofclass(Client, cli)
                then return cli.getMilestone()
                else return new Milestone()
pre (userIsClient(user) or userIsTrainer(user))
        and user <> "undefined";


/*********************************************************/
/*********************  USER  *********************/
```

```
	/*********************************************************/

	-- Returns all users
	pure public getAllUsers : () ==> set of MyUtils`MyString
	getAllUsers() == return allUsers;

	-- Returns user
	public getUser : () ==> MyUtils`MyString
	getUser() == return user;


	/*********************************************************/
	/*******************  CLIENT                  ********************/
	/*********************************************************/

	-- Checks if user is a client
	public pure userIsClient : MyUtils`MyString ==> bool
	userIsClient(cli) == (
		for all c in set clients do (
			if cli = c.getEmail()
			then return true
		);
		return false;
	);

	-- Gets client by email
	public getClientByEmail : MyUtils`MyString ==> Client
	getClientByEmail(cli) == (
		for all c in set clients do (
			if cli = c.getEmail()
			then return c;
		);
		return new Client();
	)
	pre userIsClient(cli);

	-- Gets current logged in user
	public getCurrentClient : () ==> Client
	getCurrentClient() == (
		return getClientByEmail(user);
	)
	pre user <> "undefined" and userIsClient(user);

	-- Returns all registered clients
	pure public getClients : () ==> set of Client
	getClients() == return clients;

-- Adds client to registered clients.

	public addClient : Client ==> ()
	addClient(client) == (
```

```
		atomic(
			clients := clients union {client};
			allUsers := allUsers union {client.getEmail()}
		);
)
pre ({client} inter clients = {}) and client.getEmail() not in set allUsers
post client in set clients and client.getEmail() in set allUsers;

-- Removes client from registered clients
public removeClient : Client ==> ()
removeClient(client) == (
		atomic(
			clients := clients \ {client};
			allUsers := allUsers \ {client.getEmail()}
		);
)
pre client in set clients and client.getEmail() in set allUsers
post client not in set clients and {client.getEmail()} inter allUsers = {};



/*************************************************************/
/*********************  TRAINERS  ********************/
/*************************************************************/

-- Returns all trainers
pure public getTrainers : () ==> set of Trainer
getTrainers() == return trainers;

-- Checks if user is a trainer
public pure userIsTrainer : MyUtils`MyString ==> bool
userIsTrainer(email) == (
		for all t in set trainers do (
			if email = t.getEmail()
			then return true
		);
		return false;
);

-- Adds trainer to trainers. If can't add trainer, it returns false
public addTrainer : Trainer ==> ()
addTrainer(trainer) == (
		atomic(
			trainers := trainers union {trainer};
			allUsers := allUsers union {trainer.getEmail()};
		);
)
pre trainer not in set trainers and trainer.getEmail() not in set allUsers
post trainer in set trainers;

-- Removes trainer from trainers
public removeTrainer : Trainer ==> ()
```

```
removeTrainer(trainer) ==
(
	atomic(
		trainers := trainers \ {trainer};
		allUsers := allUsers \ {trainer.getEmail()};
	);
)
pre ({trainer} subset trainers) = true and trainer.getEmail() in set allUsers
post ({trainer} subset trainers) = false;

-- Gets trainer by email
public getTrainerByEmail : MyUtils`MyString ==> Trainer
getTrainerByEmail(email) == (
	for all t in set trainers do (
		if email = t.getEmail()
		then return t;
	);
	return new Trainer();
)
pre userIsTrainer(email);



/*************************************************************/
/*********************  EXERCISES  *********************/
/*************************************************************/

-- Gets alls exercises in exercises set
public getExercises : () ==> set of Exercise
getExercises() == (
	return exercises;
);

-- Checks if exercise exists
public pure exerciseExists : MyUtils`MyString ==> bool
exerciseExists(name) == (
	for all ex in set exercises do (
		if (name = ex.getName())
		then return true;
	);
	return false;
);

-- Gets exercise in exercises set by name
public getExercise : MyUtils`MyString ==> Exercise
getExercise(name) == (
	for all ex in set exercises do (
		if (name = ex.getName())
		then return ex;
	);
	return new Exercise();
)
```

```
pre exerciseExists(name);

-- Adds exercise to exercises set
public addExercise : Exercise ==> ()
addExercise(exercise) == (
        exercises := exercises union {exercise}
)
pre not {exercise} subset exercises
post {exercise} subset exercises;

-- Removes exercise from exercises set
public removeExercise : Exercise ==> ()
removeExercise(exercise) == (
        exercises := exercises \ {exercise}
)
pre {exercise} subset exercises
post not {exercise} subset exercises;


/*********************************************************/
/******************          SERIES          *******************/
/*********************************************************/

-- Creates a series (load)
public createSeriesLoad : rat * nat1 * Exercise ==> Series
createSeriesLoad(l, r, e) == (
        return new Series(l, r, e);
);

-- Creates a series (time)
public createSeriesTime : nat1 * Exercise * nat ==> Series
createSeriesTime(r, e, t) == (
        return new Series(r, e, t);
);

/*********************************************************/
/*******************      SET     ********************/
/*********************************************************/

-- Creates a set
public createSet : () ==> MySet
createSet() == (
        return new MySet();
);

-- Add series to set
public addSeriesToSet : MySet * Series ==> ()
addSeriesToSet(s, srs) ==
        s.addSeries(srs);
```

```
/*******************************************************/
/******************       TRAINING PLAN          *******************/
/*******************************************************/

-- A trainer creates a daily plan to add to the training plan of a client
public setDailyPlan : MyUtils`MyString * nat * MySet==> ()
setDailyPlan(email, day, s) == (
        let cli = getClientByEmail(email)
        in
                (
                        cli.getMilestone().getTrainingPlan().setDailyPlan(day, s);
                )
)
pre userIsClient(email) and userIsTrainer(user);

-- A trainer consults a daily plan of the training plan of a client
public getDailyPlan : MyUtils`MyString * nat==> MySet
getDailyPlan(email, day) == (
        let cli = getClientByEmail(email)
        in
                (
                        return cli.getMilestone().getTrainingPlan().getDailyPlan(day);
                )
)
pre userIsClient(email) and ( userIsTrainer(user) or userIsClient(user));


end IronTrainers
```

Validação do modelo

Test

```
class Test
operations

        -- fails if expected is not equal to actual
        protected static assertEquals: ? * ? ==> ()
        assertEquals(expected, actual) ==
                if expected <> actual then
                (
                        IO`print("Actual value(");
                        IO`print(actual);
                        IO`print(") different from expected (");
                        IO`print(expected);
                        IO`print(")\n");
                )
        post expected = actual;

        protected static assertNotEquals: ? * ? ==> ()
        assertNotEquals(expected, actual) ==
```

```
                if expected = actual then
                (
                        IO`print("Actual value(");
                        IO`print(actual);
                        IO`print(") equal is equal to expected (");
                        IO`print(expected);
                        IO`print(") when it SHOULDN'T be.\n");
                )
        post expected <> actual;

        public static run: () ==> ()
        run() ==
        (
                UserTest`run();
                TrainerTest`run();
                ClientTest`run();

                MilestoneTest`run();

                ExerciseTest`run();
                SeriesTest`run();
                SetTest`run();
                TrainingPlanTest`run();

                IronTrainersTest`run();

                TestCases`run();
        );

end Test
```

UserTest

```
class UserTest is subclass of Test
instance variables

        testEmail: seq of char := "testEmail";
        testPassword: seq of char := "testPassword";
        testName: seq of char := "testName";
        testMobileNumber : int := 911911911;

        emptyEmail: seq of char := "";
        emptyPassword: seq of char := "";
        emptyName: seq of char := "";

        invalidMobileNumber1: int := 911;
        invalidMobileNumber2: int := 199119119;

        u1 : User := new User(testEmail, testPassword, testName);
        u2 : User := new User(testEmail, testPassword, testName, testMobileNumber);

operations

        private testConstructor: () ==> ()
        testConstructor() ==
        (
                assertEquals(u1.getEmail(), testEmail);
                assertEquals(u1.getPassword(), testPassword);
                assertEquals(u1.getName(), testName);
```

```
		assertEquals(u2.getEmail(), testEmail);
		assertEquals(u2.getPassword(), testPassword);
		assertEquals(u2.getName(), testName);
		assertEquals(u2.getMobileNumber(), testMobileNumber);
	);

	private testWithEmptyInputs: () ==> ()
	testWithEmptyInputs() ==
	(
		assertEquals(new User(emptyEmail, emptyPassword, emptyName), nil);
	);

	private testWithInvalidMobileNumber: () ==> ()
	testWithInvalidMobileNumber() ==
	(
		--assertEquals(new User(testEmail, testPassword, testName, invalidMobileNumber1),nil);
		assertEquals(new User(testEmail, testPassword, testName, invalidMobileNumber2),nil);
	);

	public static run: () ==> ()
	run() ==
	(
		new UserTest().testConstructor();

	/***** TEST CASES WITH INVALID INPUTS (EXECUTE ONE AT A TIME) ******/

		--new UserTest().testWithInvalidMobileNumber();	-- does not respect mobile number pre-condition
		--new UserTest().testWithEmptyInputs();
	);

traces
-- test cases will be generated in all possible combinations
-- must use the 'Combinatory Testing' (CT) perspective
-- calls u.getName() 1 to 5 times when selecting Full Evaluation
-- if we use 'Filtered Evaluation' we can random the number of times it is called
-- these tests do not account for coverage

	GetNameSeveralTimes:
		u1.getName(){1, 5};

end UserTest
```

ClientTest

```
class ClientTest is subclass of Test
instance variables

	-- To import a type from another classe, use Class`Type (with the '`' character).

	testEmail: seq of char := "testEmail";
	testPassword: seq of char := "testPassword";
	testName: seq of char := "testName";
	testMobileNumber : int := 911911911;
	testGender : Client`Sex := <F>;
	testGender2 : Client`Sex := <M>;
	testWeight: real := 65.0;
	testHeight: int := 180;
```

```
	testBirthDate: MyUtils`Date := mk_MyUtils`Date(2, 3, 1980);

	c1 : Client := new Client(testEmail, testPassword, testName, testGender, testWeight, testHeight,
testBirthDate);
	c2 : Client := new Client(testEmail, testPassword, testName, testGender2, testWeight, testHeight,
testBirthDate);

operations

	private testConstructor: () ==> ()
	testConstructor() ==
	(
		assertEquals(c1.getEmail(), testEmail);
		assertEquals(c1.getPassword(), testPassword);
		assertEquals(c1.getName(), testName);
		assertEquals(c1.getGender(), testGender);
		assertEquals(c1.getWeight(), testWeight);
		assertEquals(c1.getHeight(), testHeight);
		assertEquals(c1.getAge(), MyUtils`getCurrentDate().year - testBirthDate.year);
	);

	/*** USE CASE SCENARIO R4 AND R5: Edit Profile and Change Milestone ***/
	-- A user may change its information.
	-- A user may change its milestone.

	private testSetters: () ==> ()
	testSetters() ==
	(
		dcl newHeight : nat  := 2;
		dcl newWeight : real  := 1.2;
		dcl newDesiredWeight : rat := 60.0;

		c1.setWeight(newWeight);
		assertEquals(c1.getWeight(), newWeight);

		c1.setHeight(newHeight);
		assertEquals(c1.getHeight(), newHeight);

		c1.getMilestone().setDesiredWeight(newDesiredWeight);
		assertEquals(c1.getMilestone().getDesiredWeight(), newDesiredWeight);

	);

	public static run: () ==> ()
	run() ==
	(
		new ClientTest().testConstructor();
		new ClientTest().testSetters();
	);

end ClientTest
```

TrainerTest

```
class Trainer is subclass of User

operations

	public Trainer : MyUtils`MyString * MyUtils`MyString * MyUtils`MyString ==> Trainer
```

```
	Trainer(e, p, n) == (
		User(e, p, n);
	);

end Trainer
```

ExerciseTest

```
class ExerciseTest is subclass of Test
instance variables

	testName : seq of char := "testName";
	testDescription: seq of char := "testDescription";
	testBodyPart1: Exercise`BodyPart := <Arm>;
	testBodyPart2: Exercise`BodyPart := <Leg>;
	testBodyPart3: Exercise`BodyPart := <Chest>;
	testBodyPart4: Exercise`BodyPart := <Back>;

	e1 : Exercise := new Exercise(testName, testDescription, testBodyPart1);
	e2 : Exercise := new Exercise(testName, testDescription, testBodyPart2);
	e3 : Exercise := new Exercise(testName, testDescription, testBodyPart3);
	e4 : Exercise := new Exercise(testName, testDescription, testBodyPart4);


operations

	private testConstructor: () ==> ()
	testConstructor() ==
	(
		assertEquals(e1.getName(), testName);
		assertEquals(e1.getDescription(), testDescription);
		assertEquals(e1.getBodyPart(), testBodyPart1);

		assertEquals(e2.getBodyPart(), testBodyPart2);

		assertEquals(e3.getBodyPart(), testBodyPart3);

		assertEquals(e4.getBodyPart(), testBodyPart4);
	);

	public static run: () ==> ()
	run() ==
	(
		new ExerciseTest().testConstructor();
	);

end ExerciseTest
```

SeriesTest

```
class SeriesTest is subclass of Test
instance variables

	kgToLbs : rat := 2.20462262;

	testLoad: rat := 120.0;
	testTime: nat := 60;
	testRepetitions: nat1 := 5;
```

```
	testExerciseName : seq of char := "testExerciseName";
	testExerciseDescription : seq of char := "testExerciseDescription";
	testExerciseBodyPart : Exercise`BodyPart := <Arm>;

	testExercise : Exercise := new Exercise(testExerciseName, testExerciseDescription,
testExerciseBodyPart);

	s1 : Series := new Series(testLoad, testRepetitions, testExercise);
	s2 : Series := new Series(testRepetitions, testExercise, testTime);

operations

	private testConstructor: () ==> ()
	testConstructor() ==
	(
		assertEquals(s1.getLoad(), testLoad);
		assertEquals(s1.getNumberOfRepetitions(), testRepetitions);
		assertEquals(s1.getExercise(), testExercise);

		assertEquals(s2.getTime(), testTime);
		assertEquals(s2.getNumberOfRepetitions(), testRepetitions);
		assertEquals(s2.getExercise(), testExercise);
	);

	-- USE CASE R15
	-- It converts kg to lbs
	private testFunctions: () ==> ()
	testFunctions () ==
	(
		assertEquals(s1.convertFromKgToLbs(s1.getLoad()), testLoad * kgToLbs);
	);

	public static run: () ==> ()
	run() ==
	(
		new SeriesTest().testConstructor();
		new SeriesTest().testFunctions();
	);
end SeriesTest
```

SetTest

```
class SetTest is subclass of Test
instance variables

	testLoad: rat := 120.0;
	testTime: nat := 60;
	testRepetitions: nat1 := 5;

	testExerciseName : seq of char := "testExerciseName";
	testExerciseDescription : seq of char := "testExerciseDescription";
	testExerciseBodyPart : Exercise`BodyPart := <Arm>;

	testExercise : Exercise := new Exercise(testExerciseName, testExerciseDescription,
testExerciseBodyPart);

	testSeries1 : Series := new Series(testLoad, testRepetitions, testExercise);
	testSeries2 : Series := new Series(testRepetitions, testExercise, testTime);
```

```
        testSeries3 : Series := new Series(testLoad + 10.0, testRepetitions + 5, testExercise);
        testSeries4 : Series := new Series(testRepetitions + 10, testExercise, testTime + 20);

        testSeriesSequence : seq of Series := [testSeries1, testSeries2, testSeries3];

        s1 : MySet := new MySet(testSeriesSequence);
        s2 : MySet := new MySet();

operations

        private testConstructor: () ==> ()
        testConstructor() ==
        (
                assertNotEquals(s1.getIndex(testSeries1), -1);
                assertNotEquals(s1.getIndex(testSeries2), -1);
                assertNotEquals(s1.getIndex(testSeries3), -1);

                assertEquals(s1.getSeries(), testSeriesSequence);

                assertEquals(s1.getIndex(testSeries4), -1);

                assertEquals(len s2.getSeries(), 0);
        );

        private testAddSeries: () ==> ()
        testAddSeries () ==
        (
                s1.addSeries(testSeries4);
                assertNotEquals(s1.getIndex(testSeries4), -1);
        );

        public static run: () ==> ()
        run() ==
        (
                new SetTest().testConstructor();
                new SetTest().testAddSeries();
        );

        public getTestSet: () ==> MySet
        getTestSet() ==
        (
                return s1;
        );

end SetTest
```

TrainingPlanTest

```
class TrainingPlanTest is subclass of Test
instance variables

        exampleSet: MySet := new SetTest().getTestSet();

        tp1 : TrainingPlan := new TrainingPlan();

operations

        private testConstructor: () ==> ()
        testConstructor() ==
```

```
	(
		assertEquals(tp1.getPlan(), {|->});
	);


	private testGetters: () ==> ()
	testGetters() ==
	(
		tp1.setDailyPlan(1, exampleSet);
		assertEquals(tp1.getDailyPlan(1), exampleSet);
	);

	private testSetters: () ==> ()
	testSetters() ==
	(
		tp1.setDailyPlan(1, exampleSet);
		assertEquals(tp1.getDailyPlan(1), exampleSet);
	);

	public static run: () ==> ()
	run() ==
	(
		new TrainingPlanTest().testConstructor();
		new TrainingPlanTest().testGetters();
		new TrainingPlanTest().testSetters();
	);

end TrainingPlanTest
```

MilestoneTest

```
class MilestoneTest is subclass of Test
instance variables

	testDesiredWeight : rat := 60.0;

	m1 : Milestone := new Milestone(testDesiredWeight);

operations

	private testConstructor: () ==> ()
	testConstructor() ==
	(
		assertEquals(m1.getDesiredWeight(), testDesiredWeight);
	);

	private testSetters: () ==> ()
	testSetters() ==
	(
		dcl testNewDesiredWeight : rat := 65.0;

		m1.setDesiredWeight(testNewDesiredWeight);
		assertEquals(m1.getDesiredWeight(), testNewDesiredWeight);
	);

	public static run: () ==> ()
	run() ==
	(
		new MilestoneTest().testConstructor();
```

```
		new MilestoneTest().testSetters();
	);
end MilestoneTest
```

TestCases

```
class TestCases is subclass of Test

/*
* Class which includes all test cases for the mandatory requirements of the app.
*/

operations

/*** USE CASE SCENARIO R0: Initial configuration ***/
-- The app initially has no clients and only one registered trainer account.

	private initialConfigScn : Trainer ==> IronTrainers
	initialConfigScn(trainer) ==
	(
		dcl it : IronTrainers := new IronTrainers();
		it.addTrainer(trainer);
		return it;
	)
	post (RESULT.getTrainers() = {trainer} and-- only the trainer is registered as trainer
		RESULT.getClients() = {} and		-- no clients are registered
		card RESULT.getAllUsers() = 1 and		-- there is only one user (the
trainer)
		RESULT.getAllUsers() = {trainer.getEmail()}); -- confirm that the sole
user's email is the trainer email

/*** USE CASE SCENARIO R1: Register ***/
-- A user may register as a Client using an email and a password, and providing the profile information.


	public static run: () ==> ()
	run() ==
	(
		dcl it0 : IronTrainers := new TestCases().initialConfigScn(new Trainer("diogo@gmail.com",
"pass123", "Diogo"));


		/* These asserts are only for suppressing warnings about the variables not being used.*/
		assertEquals(it0, it0);
	);

end TestCases
```

IronTrainersTest

```
class IronTrainersTest is subclass of Test
instance variables

	e1 : IronTrainers := new IronTrainers();

operations

	private testConstructor: () ==> ()
```

```
testConstructor() ==
(
        assertEquals(e1.getClients(), {});
        assertEquals(e1.getTrainers(), {});

);

/*** USE CASE SCENARIO R1: Register ***/
-- A user may register as a Client using an email and a password, and providing the profile information.

private testCreateClient : () ==> Client
testCreateClient() ==
(
        dcl testEmail: seq of char := "testEmail";
        dcl testPassword: seq of char := "testPassword";
        dcl testName: seq of char := "testName";
        dcl testGender : Client`Sex := <F>;
        dcl testWeight: real := 65.0;
        dcl testHeight: int := 180;
        dcl testBirthDate: MyUtils`Date := mk_MyUtils`Date(2, 3, 1980);

        dcl c1 : Client := new Client(testEmail, testPassword, testName, testGender, testWeight,
testHeight, testBirthDate);

        return c1;
);

private testCreateTrainer : () ==> Trainer
testCreateTrainer() ==
(
        dcl testEmail: seq of char := "testEmailTrainer";
        dcl testPassword: seq of char := "testPassword";
        dcl testName: seq of char := "testNameTrainer";

        dcl t1 : Trainer := new Trainer(testEmail, testPassword, testName);

        return t1;
);

private testAddExercises : () ==> ()
testAddExercises() ==
(
        dcl testName: MyUtils`MyString := "Running";
        dcl testName2: MyUtils`MyString := "Squats";
        dcl testName3: MyUtils`MyString := "Jumps";
        dcl testName4: MyUtils`MyString := "Cycling";
        dcl testDescription: MyUtils`MyString := "Running";
        dcl testBodyPart1: Exercise`BodyPart := <Leg>;
        dcl testBodyPart2: Exercise`BodyPart := <Arm>;
        dcl testBodyPart3: Exercise`BodyPart := <Chest>;
        dcl testBodyPart4: Exercise`BodyPart := <Back>;

        dcl ex1 : Exercise := new Exercise(testName, testDescription, testBodyPart1);
        dcl ex2 : Exercise := new Exercise(testName2, testDescription, testBodyPart2);
        dcl ex3 : Exercise := new Exercise(testName3, testDescription, testBodyPart3);
        dcl ex4 : Exercise := new Exercise(testName4, testDescription, testBodyPart4);

        -- Add exercises
        e1.addExercise(ex1);
        assertEquals(e1.getExercises(), {ex1});
```

```
		e1.addExercise(ex2);
		assertEquals(e1.getExercises(), {ex1, ex2});

		e1.addExercise(ex3);
		assertEquals(e1.getExercises(), {ex1, ex2, ex3});

		e1.addExercise(ex4);
		assertEquals(e1.getExercises(), {ex1, ex2, ex3, ex4});
);

private testClient: () ==> ()
testClient() ==
(
		dcl c1 : Client := testCreateClient();

		e1.addClient(c1);

		assertEquals(e1.getClients(), {c1});
		assertEquals(e1.getAllUsers(), {c1.getEmail()});

		e1.removeClient(c1);

		assertEquals(e1.getClients(), {});
		assertEquals(e1.getAllUsers(), {});

);

private testTrainer: () ==> ()
testTrainer() ==
(
		dcl t1 : Trainer := testCreateTrainer();

		e1.addTrainer(t1);

		assertEquals(e1.getTrainers(), {t1});
		assertEquals(e1.getAllUsers(), {t1.getEmail()});

		e1.removeTrainer(t1);

		assertEquals(e1.getTrainers(), {});
		assertEquals(e1.getAllUsers(), {});

);

/*** USE CASE SCENARIO R2: Client Login ***/
-- A user may enter as a Client using an email and a password.

private testClientLoginLogout: () ==> ()
testClientLoginLogout() ==
(
		dcl testEmail: seq of char := "testEmail";
		dcl testPassword: seq of char := "testPassword";

		dcl c1 : Client := testCreateClient();

		e1.addClient(c1);

		-- Login
		assertEquals(e1.loginClient(testEmail, testPassword), true);
```

```
		assertEquals(e1.getUser(), testEmail);
		assertEquals(e1.getCurrentClient(), c1);

		-- Logout
		e1.logout();
		assertEquals(e1.getUser(), "undefined");
);

/*** USE CASE SCENARIO R2: Trainer Login ***/
-- A user may enter as a Trainer using an email and a password.

private testTrainerLoginLogout: () ==> ()
testTrainerLoginLogout() ==
(
		dcl testEmail: seq of char := "testEmailTrainer";
		dcl testPassword: seq of char := "testPassword";

		dcl t1 : Trainer := testCreateTrainer();

		e1.addTrainer(t1);

		-- Login
		assertEquals(e1.loginTrainer(testEmail, testPassword), true);
		assertEquals(e1.getUser(), testEmail);

		-- Logout
		e1.logout();
		assertEquals(e1.getUser(), "undefined");
);

private testNotRegisterLoginLogout: () ==> ()
testNotRegisterLoginLogout() ==
(
		dcl testEmail: seq of char := "testEmail";
		dcl testPassword: seq of char := "testPassword";

		-- Login
		assertEquals(e1.loginClient(testEmail, testPassword), false);
		assertEquals(e1.getUser(), "undefined");

		-- Logout
		e1.logout();
		assertEquals(e1.getUser(), "undefined");
);

private testDuplicateTrainerEmails: () ==> ()
testDuplicateTrainerEmails() ==
(
		dcl testEmail: seq of char := "testEmail";
		dcl testPassword: seq of char := "testPassword";
		dcl testName: seq of char := "testName";

		dcl t1 : Trainer := new Trainer(testEmail, testPassword, testName);
		dcl t2 : Trainer := new Trainer(testEmail, testPassword, testName);

		e1.addTrainer(t1);
		e1.addTrainer(t2);
);

private testDuplicateUserEmails: () ==> ()
```

```
	testDuplicateUserEmails() ==
	(
		dcl testEmail: seq of char := "testEmail";
		dcl testPassword: seq of char := "testPassword";
		dcl testName: seq of char := "testName";
		dcl testGender : Client`Sex := <F>;
		dcl testWeight: real := 65.0;
		dcl testHeight: int := 180;
		dcl testBirthDate: MyUtils`Date := mk_MyUtils`Date(2, 3, 1980);

		dcl c1 : Client := new Client(testEmail, testPassword, testName, testGender, testWeight,
testHeight, testBirthDate);

		dcl t1 : Trainer := new Trainer(testEmail, testPassword, testName);

		e1.addTrainer(t1);
		e1.addClient(c1);
	);

	private testCurrentYear: () ==> ()
	testCurrentYear() ==
	(
		assertEquals(MyUtils`getCurrentDate(), mk_MyUtils`Date(3, 1, 2018));
	);

	private testMilestone: () ==> ()
	testMilestone() ==
	(
		dcl testEmail: seq of char := "testEmail";
		dcl testEmail2: seq of char := "testEmailTrainer";
		dcl testPassword: seq of char := "testPassword";

		dcl testDesiredWeight: real := 60.0;

		dcl c1 : Client := testCreateClient();
		dcl t1 : Trainer := testCreateTrainer();

		e1.addClient(c1);
		e1.addTrainer(t1);

		-- Login with client account
		assertEquals(e1.loginClient(testEmail, testPassword), true);

		-- Update Milestone
		e1.updateDesiredWeight(testDesiredWeight);
		assertEquals(e1.getCurrentClient(), c1);
		assertEquals(e1.consultMilestone(e1.getUser()).getDesiredWeight(), testDesiredWeight);

		e1.logout();

		-- Login with trainer account
		assertEquals(e1.loginTrainer(testEmail2, testPassword), true);

		-- Consult Training plan
		assertEquals(e1.consultMilestone(testEmail).getTrainingPlan().getPlan(), { |-> });

		-- Add training Plan
		assertEquals(e1.consultMilestone(testEmail).getTrainingPlan().getPlan(), { |-> });

	);
```

```
	private testExercise: () ==> ()
	testExercise() ==
	(
		dcl testName: MyUtils`MyString := "Running";
		dcl testDescription: MyUtils`MyString := "Running";
		dcl testBodyPart1: Exercise`BodyPart := <Leg>;
		dcl testBodyPart2: Exercise`BodyPart := <Arm>;
		dcl testBodyPart3: Exercise`BodyPart := <Chest>;
		dcl testBodyPart4: Exercise`BodyPart := <Back>;

		dcl ex1 : Exercise := new Exercise(testName, testDescription, testBodyPart1);
		dcl ex2 : Exercise := new Exercise(testName, testDescription, testBodyPart2);
		dcl ex3 : Exercise := new Exercise(testName, testDescription, testBodyPart3);
		dcl ex4 : Exercise := new Exercise(testName, testDescription, testBodyPart4);

		-- Add exercises
		e1.addExercise(ex1);
		assertEquals(e1.getExercises(), {ex1});

		e1.addExercise(ex2);
		assertEquals(e1.getExercises(), {ex1, ex2});

		e1.addExercise(ex3);
		assertEquals(e1.getExercises(), {ex1, ex2, ex3});

		e1.addExercise(ex4);
		assertEquals(e1.getExercises(), {ex1, ex2, ex3, ex4});

		-- Remove exercises
		e1.removeExercise(ex4);
		assertEquals(e1.getExercises(), {ex1, ex2, ex3});

		e1.removeExercise(ex3);
		assertEquals(e1.getExercises(), {ex1, ex2});

		e1.removeExercise(ex2);
		assertEquals(e1.getExercises(), {ex1});

		e1.removeExercise(ex1);
		assertEquals(e1.getExercises(), {});

	);

	/*** USE CASE SCENARIO R6:  Client can consult its training plan ***/
	-- A client can consult the training plan related to its milestone
	/*** USE CASE SCENARIO R8, R9, R10, R11, R12, R13 and R14: Trainer associates a training plan
to a milestone of a client ***/
	-- Trainer adds Exercise
	-- Trainer creates Series
	-- Trainer creates Set
	-- Trainer creates TrainingPlan
	-- Trainer associates TrainingPlan to the Milestone of the Client


	private testDailyPlan: () ==> ()
	testDailyPlan() ==
	(
		dcl s1 : MySet := e1.createSet();
```

```
		dcl s2 : MySet := e1.createSet();
		dcl series1 : Series;
		dcl series2 : Series;
		dcl ex1 : Exercise;
		dcl ex2 : Exercise;

		dcl testEmail: seq of char := "testEmail";
		dcl testEmailTrainer: seq of char := "testEmailTrainer";
		dcl testPassword: seq of char := "testPassword";


		dcl c1 : Client := testCreateClient();
		dcl t1 : Trainer := testCreateTrainer();

		e1.addClient(c1);
		e1.addTrainer(t1);

		testAddExercises();

		-- Login
		assertEquals(e1.loginTrainer(testEmailTrainer, testPassword), true);

		-- Create MySet Load
		ex1 := e1.getExercise("Running");
		series1 := e1.createSeriesTime(1, ex1, 60);

		e1.addSeriesToSet(s1, series1);

		e1.setDailyPlan(testEmail, 1, s1);
		assertEquals(e1.getDailyPlan(testEmail, 1), s1);

		-- Create MySet Time
		ex2 := e1.getExercise("Squats");
		series2 := e1.createSeriesLoad(10.0, 10, ex2);

		e1.addSeriesToSet(s2, series2);

		e1.setDailyPlan(testEmail, 2, s2);
		assertEquals(e1.getDailyPlan(testEmail, 2), s2);

		e1.logout();

		-- Login with client account
		assertEquals(e1.loginClient(testEmail, testPassword), true);
		assertEquals(e1.getDailyPlan(testEmail, 2), s2);

);

public static run: () ==> ()
run() ==
(
		new IronTrainersTest().testConstructor();
		new IronTrainersTest().testClient();
		new IronTrainersTest().testTrainer();
		new IronTrainersTest().testCurrentYear();
		new IronTrainersTest().testClientLoginLogout();
		new IronTrainersTest().testTrainerLoginLogout();
		new IronTrainersTest().testMilestone();
		new IronTrainersTest().testExercise();
		new IronTrainersTest().testDailyPlan();
```

```
    /***** TEST CASES WITH INVALID INPUTS (EXECUTE ONE AT A TIME) ******/

                --new IronTrainersTest().testNotRegisterLoginLogout();
                --new IronTrainersTest().testDuplicateTrainerEmails();
                --new IronTrainersTest().testDuplicateUserEmails();
        );

end IronTrainersTest
```

Verificação do modelo

Exemplo de verificação de domínio

Uma das *Proof Obligations* geradas pelo Overture é:

Número	Nome	Tipo
10	IronTrainers`addClient(Client)	Adição de um elemento a um *Set*.

O código sob análise é o seguinte:

```
-- Adds client to registered clients.
public addClient : Client ==> ()
addClient(client) == (
    atomic(
        clients := clients union {client};
        allUsers := allUsers union {client.getEmail()}
    );
)
pre ({client} inter clients = {}) and client.getEmail() not in
set allUsers
post client in set clients and client.getEmail() in set allUsers;
```

Esta operação adiciona um *Client* ao *set* de *Clients* de um objeto *IronTrainers*. Para evitar duplicações de identificadores, que neste caso é o email de cada cliente, o domínio desta operação é restrito a:

- Objetos *Client* que não existam já no conjunto de clientes:
  - {client} inter clients = {}
- Objetos *Client* cujos emails não estejam atribuídos a nenhum utilizador, quer seja ele *Client* ou *Trainer*:
  - client.getEmail() not in set allUsers

A segunda condição é uma versão mais forte da primeira. Por exemplo, existindo um cliente já registado com o email "email@domain.com" e tentando um novo cliente registar-se com esse email, o programa não o permitiria. Admitindo que um cliente é caracterizado apenas pelo seu email, podemos elaborar a seguinte prova:

1. c1.email = "email@domain.com"
2. clients = {c1}
3. newClient = "email@domain.com"
4. clients inter newClient = {c1}
5. {c1} != {}

Assim, a primeira condição já falharia e o cliente não seria registado, pois já existia um cliente com a sua informação toda duplicada. No entanto, esta condição deixava que clientes se pudessem registar com emails que já tinham sido atribuídos a treinadores, pelo que foi necessário acrescentar a segunda condição.

1. t1.email = email@domain.com
2. trainers = {t1}
3. clients = {}
4. allUsers = {“email@domain.com”}
5. newClient = “email@domain.com”
6. clients inter newClient = {}, pelo que a primeira condição era verdadeira
7. “email@domain.com” not in {“email@domain.com”}

A condição no último ponto é falsa, pelo que o utilizador não seria registado.

A *proof obligation* gerada pelo Overture é a seguinte:

```
(forall client:Client &
        (((({client} inter clients) = {}) and
        ((client.getEmail)() not in set allUsers)) =>

        ((client in set (clients union {client})) and
        ((client.getEmail)() in set (allUsers union{(client.getEmail)()}))))
```

Exemplo de verificação de invariante

Outra das *Proof Obligations* geradas pelo Overture é:

Número	Nome	Tipo
4	Client`setWeight(real)	Estado da invariante é mantido.

O código sob análise é o seguinte:

```
-- Set weight
    public setWeight : real ==> ()
    setWeight(w) == weight := w
    pre not w <= 0;
```

Esta operação altera o valor do peso de um cliente para um novo valor real. Na declaração da variável de instância “weight” é especificada a seguinte variância:

```
inv weight > 0 and height > 0;
```

Esta variância impede que valores inválidos de altura e de massa sejam inseridos na aplicação, já que é impossível que a altura e a massa sejam valores negativos.

Após a execução do bloco, o valor “weight” da instância de *Client* é alterado para um determinado número real que tenha sido dado como argumento. No entanto, a pré condição:

```
pre not w <= 0;
```

Apenas permite que valores positivos sejam utilizados como argumento. Desta forma, a invariância mantém-se.

A *proof obligation* gerada pelo Overture é a seguinte:

```
(forall w:real &
        ((not (w <= 0)) => (((weight > 0) and (height > 0)) =>
        ((w > 0) and (height > 0)))))
```

Geração de Código

A partir do modelo elaborado no VDM++, foi gerado código Java. A partir desse mesmo código, foi criada uma interface, que se encontra no package ***gui***, para facilitar a demonstração do trabalho que foi desenvolvido. Esta interface recorre às classes do package ***model*** geradas pelo VDM++.

Conclusões

O modelo desenvolvido cobre todos os requisitos.

Este projeto demorou cerca de 25 horas por pessoa a concluir. Perto de um terço deste período foi dedicado à elaboração do modelo, recorrendo a diagramas e implementando utilizando a ferramenta Overture. A elaboração de testes ao modelo e respetiva correção ocupou perto de 10 horas, enquanto que a implementação da interface gráfica e elaboração deste relatório ocupou a restante porção de tempo.

Referências

1. Site do Overture, http://overturetool.org
2. “Report Vending Machine”, J. P. Faria
3. “VDM++”, A. C. Paiva