MFES

January 6, 2019

Contents

1 Date

```
class Data
types
 public Date :: day : nat
          month : nat
          year : nat
  inv d == d.year > 2018 and
    d.month <= 12 and
    d.day <=  DaysOfMonth(d.year, d.month);
values
-- TODO Define values here
instance variables

operations
-- TODO Define operations here
functions

 public DaysOfMonth : nat * nat -> nat
 DaysOfMonth(y, m) ==
```

```
  if (m = 2) then (
   if isLeapYear(y) then 29
   else 28
  )
  else if (m = 4 or m = 6 or m = 9 or m = 11) then 30
  else 31;


 public isLeapYear : nat -> bool
 isLeapYear (y) ==
  y mod 4 = 0 and y mod 100 <> 0 or y mod 400 = 0;
traces
-- TODO Define Combinatorial Test Traces here
end Data
```

2 Event

```
class Event is subclass of Data
types
-- TODO Define types here
  public String = seq of char;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private name : String;
 private date : Date;
 private capacity : nat;
 private popularity: nat := 0;
 private tickets : set of nat := {};
  inv card tickets <= capacity;
 private ticketPrice : rat;
operations
-- TODO Define operations here

 public Event : String * nat * rat * Date ==> Event
 Event(n, c, p, d) == (
  name := n;
  capacity := c;
  ticketPrice := p;
  date := d;
  return self
 )
 pre p > 0
 post name = n and capacity = c and tickets = {} and popularity = 0;


 public pure getName : () ==> String
 getName()==(
  return name;
 );


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


 public pure getTicketPrice : () ==> nat
```

```
 getTicketPrice()==(
  return ticketPrice;
 );


 public pure getFillPercent : () ==> real
 getFillPercent()==(
  return (card (tickets) / capacity ) * 100;
 );


 public pure getTickets : () ==> set of nat
 getTickets() == (
  return tickets;
 );


 public addTicket : nat ==> ()
 addTicket(ticket) == (
  tickets := tickets union {ticket}
 )
 pre card tickets <= capacity and ticket not in set tickets
 post tickets = tickets~ union {ticket};


 public removeTicket : nat ==> ()
 removeTicket(i) == (
  tickets:= tickets \ {i};
 )
 pre i in set tickets
 post tickets = tickets~ \ {i};


 public promote : () ==> ()
 promote() == (
  popularity := popularity + 10;
 )
 pre popularity + 10 <= 100;


 public getPopularity : () ==> nat
 getPopularity() == (
  return popularity;
 );


 public getEarnings : () ==> rat
 getEarnings() == (
  return (card tickets) * ticketPrice;
 );


 public getDate : () ==> Date
 getDate() == (
  return date;
 );

functions
-- TODO Define functiones here

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

3 EventManager

```
class EventManager
types
-- TODO Define types here
 public String = seq of char;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private events : map String to Event := {|->};
operations
-- TODO Define operations here

 public EventManager : map String to Event ==> EventManager
 EventManager(evs) == (
  events := evs;
  return self;
 )
 post events = evs;


 public pure getEvents : () ==> map String to Event
 getEvents() == (
  return events;
 );


 public pure getEvent : String ==> Event
 getEvent (e) == (
  return events(e);
 )
 pre e in set dom events;


 public addEvent : Event ==> ()
 addEvent(event) == (
  events := events ++ {event.getName() |-> event}
 )
 pre event.getName() not in set dom events
 post events = events~ ++ {event.getName() |-> event};


 public removeEvent : String ==> ()
 removeEvent(e) == (
  events:= {e} <-: events;
 )
 pre e in set dom events
 post events = {e} <-: events~;


 private pure eventExists : String ==> bool
 eventExists(e) == (return e in set dom events);


 public pure getEventTickets : String ==> set of nat
 getEventTickets(e) == (return events(e).getTickets())
 pre eventExists(e);


 public getEventTicketsUser : String*String*TicketManager ==> set of nat
 getEventTicketsUser(e,u,tm) == (
  dcl tickets : set of nat := {};
```

```
  for all ticket in set (events(e).getTickets()) do
  (
   if tm.getTickets()(ticket).getOwner() = u
    then tickets := tickets union {ticket};
  );
  return tickets;
 );


 public getEventFillPercent : String ==> rat
 getEventFillPercent(e) == (return events(e).getFillPercent();)
 pre eventExists(e);


 public addTicket : nat*String ==> ()
 addTicket(t,e) == (
  events(e).addTicket(t);
 )
 pre eventExists(e);


 public getEarnings : String ==> rat
 getEarnings(e) == (return events(e).getEarnings();)
 pre eventExists(e);


functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end EventManager
```

4 EventTest

```
class EventTest is subclass of Data
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 e : Event := new Event("evento1", 15, 10, mk_Date(10,1,2019));
 e2 : Event := new Event("evento2", 15, 10, mk_Date(29,2,2020));
 e3 : Event := new Event("evento3", 15, 10, mk_Date(29,4,2020));
 e4 : Event := new Event("evento4", 15, 10, mk_Date(28,2,2021));
 u : User := new User("dank", "memes");
 em : EventManager := new EventManager({ "evento1" |-> e, "evento2" |-> e2, "evento3" |-> e3, "
     evento4" |-> e4});
 um : UserManager := new UserManager({"dank" |-> u});
 tm : TicketManager := new TicketManager({|->});
operations
-- TODO Define operations here

 private assertTrue: bool ==> ()
  assertTrue(cond) == return
  pre cond;


 private testGetDate: () ==> ()
 testGetDate() == (
```

```
 assertTrue(e.getDate() = mk_Date(10,1,2019));
);


private testGetName: () ==> ()
testGetName() == (
 assertTrue(e.getName() = "evento1");
);


private testCapacity: () ==> ()
testCapacity() == (
 assertTrue(e.getCapacity() = 15);
);


private testGetTicketPrice: () ==> ()
testGetTicketPrice() == (
 assertTrue(e.getTicketPrice() = 10);
);


private testGetTickets : () ==> ()
testGetTickets() == (
 u.addFunds(100);
 assertTrue(um.login("dank", "memes"));
 um.buyTicket("evento1",tm,em);
 um.buyTicket("evento1",tm,em);
 assertTrue(e.getTickets() = {6,7});
);


private testRemoveTicket : () ==> ()
testRemoveTicket() == (
 u.addFunds(100);
 assertTrue(um.login("dank", "memes"));
 um.buyTicket("evento1",tm,em);
 e.removeTicket(8);
 assertTrue(e.getTickets() = {});
);


private testGetFillPercent : () ==> ()
testGetFillPercent() == (
 u.addFunds(100);
 assertTrue(um.login("dank", "memes"));
 um.buyTicket("evento1",tm,em);
 assertTrue(em.getEventFillPercent("evento1") = (1 / 15) * 100);
);


private testGetEarnings : () ==> ()
testGetEarnings() == (
 u.addFunds(100);
 assertTrue(um.login("dank", "memes"));
 um.buyTicket("evento1",tm,em);
 assertTrue(em.getEarnings("evento1") = 10);
);


private testPromotion : () ==> ()
testPromotion() == (
 e.promote();
 assertTrue(e.getPopularity() = 10);
);
```

```
 private testManagerGetEvent : () ==> ()
 testManagerGetEvent() == (
  assertTrue(em.getEvent("evento1") = e);
 );


 private testManagerAddEvent : () ==> ()
 testManagerAddEvent() == (
  dcl eventTest : Event := new Event("eventTest",10,10,mk_Date(1,1,2021));
  em.addEvent(eventTest);
  assertTrue(em.getEvent("eventTest") = eventTest);
 );


 private testManagerRemoveEvent : () ==> ()
 testManagerRemoveEvent() == (
  dcl eventTest : Event := new Event("eventTest",10,10,mk_Date(1,1,2021));
  em.addEvent(eventTest);
  em.removeEvent("eventTest");
  assertTrue(card dom em.getEvents() = 4);
 );


 private testManagerGetEventTickets : () ==> ()
 testManagerGetEventTickets() == (
  u.addFunds(100);
  assertTrue(um.login("dank", "memes"));
  um.buyTicket("evento1",tm,em);
  um.buyTicket("evento1",tm,em);
  assertTrue(card em.getEventTickets("evento1") = 2);
 );


 private testManagerGetEventTicketsUser : () ==> ()
 testManagerGetEventTicketsUser() == (
  u.addFunds(100);
  assertTrue(um.login("dank", "memes"));
  um.buyTicket("evento1",tm,em);
  um.buyTicket("evento1",tm,em);
  um.buyTicket("evento2",tm,em);
  assertTrue(card em.getEventTicketsUser("evento1","dank",tm) = 2);
 );


 public static main: () ==> ()
 main() == (
  new EventTest().testGetDate();
  new EventTest().testGetName();
  new EventTest().testCapacity();
  new EventTest().testGetTicketPrice();
  new EventTest().testGetTickets();
  new EventTest().testRemoveTicket();
  new EventTest().testGetFillPercent();
  new EventTest().testGetEarnings();
  new EventTest().testPromotion();
  new EventTest().testManagerGetEvent();
  new EventTest().testManagerAddEvent();
  new EventTest().testManagerRemoveEvent();
  new EventTest().testManagerGetEventTickets();
  new EventTest().testManagerGetEventTicketsUser();
 );
functions
-- TODO Define functiones here
```

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

5 MainTest

```
class MainTest
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
operations
-- TODO Define operations here


 public static main: () ==> ()
 main() == (
  new TicketTest().main();
  new EventTest().main();
  new UserTest().main();
 );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end MainTest
```

6 Ticket

```
class Ticket
types
-- TODO Define types here
 public String = seq of char;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private owner : String;
 private event : String;
 private id: nat;
 static count : nat := 0;
operations
-- TODO Define operations here

 public Ticket: String * String ==> Ticket
 Ticket(o,e) == (
  owner := o;
  event := e;
  id := count;
  count := count + 1;
  return self;
 )
 pre count >= 0
```

```
 post owner = o and event = e and id = count - 1;


 public pure getOwner : () ==> String
 getOwner() == (
  return owner;
 );


 public pure getEvent : () ==> String
 getEvent() == (
  return event;
 );


 public pure getID : () ==> nat
 getID() == (
  return id;
 );

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end Ticket
```

7 TicketManager

```
class TicketManager
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private tickets : map nat to Ticket := {|->};
operations
-- TODO Define operations here

 public TicketManager : map nat to Ticket ==> TicketManager
  TicketManager(ts) == (
   tickets := ts;
   return self;
  )
 post tickets = ts;


 public pure getTickets : () ==> map nat to Ticket
 getTickets() == (
  return tickets;
 );


 public addTicket : Ticket ==> ()
 addTicket(ticket) == (
  tickets := tickets ++ {ticket.getID() |-> ticket}
 )
 pre ticket.getID() not in set dom tickets
 post tickets = tickets~ ++ {ticket.getID() |-> ticket};
```

```
 public removeTicket : nat ==> ()
 removeTicket(i) == (
  tickets:= {i} <-: tickets;
 )

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TicketManager
```

8 TicketTest

```
class TicketTest is subclass of Data
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 e : Event := new Event("evento1", 15, 10, mk_Date(21,2,2019));
 u : User := new User("dank", "memes");
 em : EventManager := new EventManager({ "cenas" |-> e });
 um : UserManager := new UserManager({"dank" |-> u});
 tm : TicketManager := new TicketManager({|->});
 t : Ticket := new Ticket("dank", "evento1");
operations
-- TODO Define operations here

 private assertTrue: bool ==> ()
  assertTrue(cond) == return
  pre cond;


 private testGetID : () ==> ()
 testGetID() == (
  assertTrue(t.getID() = 1);
 );


 private testGetOwner : () ==> ()
 testGetOwner() == (
  assertTrue(t.getOwner() = "dank");
 );


 private testGetEvent : () ==> ()
 testGetEvent() == (
  assertTrue(t.getEvent() = "evento1");
 );


 private testRemoveTicket : () ==> ()
 testRemoveTicket() == (
  dcl previousCard : nat := card dom tm.getTickets();
  dcl ticketTest : Ticket := new Ticket("dank", "evento3");
  tm.addTicket(ticketTest);
  tm.removeTicket(ticketTest.getID());
  assertTrue(card dom tm.getTickets() = previousCard);
```

```
 );


 public static main: () ==> ()
 main() == (
  new TicketTest().testGetID();
  new TicketTest().testGetOwner();
  new TicketTest().testGetEvent();
  new TicketTest().testRemoveTicket();
 );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end TicketTest
```

9 User

```
class User
types
-- TODO Define types here
  public String = seq of char;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 private name : String;
 private funds : nat := 0;
 private password : String;
 private events : set of String := {};
 private tickets : set of nat := {};
operations
-- TODO Define operations here

 public User: String * String ==> User
 User(n,p) == (
  name := n;
  password := p;
  return self;
 )
 post name = n and password = p and tickets = {} and events = {};


 public pure getEvents : () ==> set of String
 getEvents() == (return events;);


 public addEvent: String ==> ()
 addEvent(e) == (events := events union {e})
 pre e not in set events;


 public removeEvent : String ==> ()
 removeEvent(e) == (events := events \ {e})
 pre e in set events;


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

```
 );


 public pure getFunds : () ==> nat
 getFunds() == (
  return funds;
 );


 public pure getPassword : () ==> String
 getPassword() == (
  return password;
 );


 public addFunds : nat ==> ()
 addFunds(f) == (
  funds := funds + f;
 )
 pre funds + f <= 10000
 post funds = funds~ + f;


 public removeFunds : nat ==> ()
 removeFunds(f) == (
  funds := funds - f;
 )
 pre funds - f >= 0
 post funds = funds~ - f;


 public pure getTickets : () ==> set of nat
  getTickets() == (
  return tickets;
 );


 public buyTicket : nat * nat ==> ()
 buyTicket(ticketPrice,t) == (
  removeFunds(ticketPrice);
  addTicket(t);
 );


 public addTicket : nat ==> ()
 addTicket(ticket) == (
  tickets := tickets union {ticket};
 )
 pre ticket not in set tickets
 post tickets = tickets~ union {ticket};


 public removeTicket : nat ==> ()
 removeTicket(i) == (
  tickets:= tickets \ {i};
 )
 pre i in set tickets
 post tickets = tickets~ \ {i};

functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end User
```

10 UserManager

```
class UserManager
types
-- TODO Define types here
 public String = seq of char;
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
  private current_user : String := "";
 private users : map String to User; -- id to User
operations
-- TODO Define operations here

 public UserManager : map String to User ==> UserManager
 UserManager(uss) == (
  users := uss;
  return self;
 );


 public login : String * String ==> bool
 login(u,p) == (
  if users(u).getPassword() = p
  then (current_user:= u; return true)
  else return false
 )
 pre u in set dom users;


 public logout : () ==> ()
 logout() == (
  current_user := ""
 )
 pre isLoggedIn();


 public register : String * String ==> ()
 register(u,p) == (
  users := users ++ {u |-> new User(u,p)};
 )
 pre u not in set dom users;


 public addUser : User ==> ()
 addUser(user) == (
  users := users ++ {user.getName() |-> user}
 )
 pre user.getName() not in set dom users
 post users = users~ ++ {user.getName() |-> user};


 public pure getUser : String ==> User
 getUser(u) == (return users(u);)
 pre u in set dom users;


 public pure getCurrentUser : () ==> String
 getCurrentUser() == (return current_user;);


 public pure getUsers : () ==> map String to User
```

```
 getUsers() == (return users;);


 private pure isLoggedIn : () ==> bool
 isLoggedIn() == (return current_user <> "" ;);


 public buyTicket : String * TicketManager * EventManager ==> ()
 buyTicket(e,tm,em) == (
 dcl ticket : Ticket := new Ticket(current_user, em.getEvents()(e).getName());
 tm.addTicket(ticket);
 em.addTicket(ticket.getID(), e);
 users(current_user).buyTicket(em.getEvents()(e).getTicketPrice(), ticket.getID()))
 pre isLoggedIn();


 public pure getUserTickets :() ==> set of nat
 getUserTickets() == (return users(current_user).getTickets())
 pre isLoggedIn();


 public getUserTicketsEvent : String * TicketManager ==> set of nat
 getUserTicketsEvent(e,tm) == (
  dcl tickets : set of nat := {};
  for all ticket in set (users(current_user).getTickets()) do
  (
   if tm.getTickets()(ticket).getEvent() = e
    then tickets := tickets union {ticket};
  );
  return tickets;
 );


 public promoteEvent : String*EventManager ==> ()
 promoteEvent(e,em) == (
  em.getEvents()(e).promote();
  users(current_user).removeFunds(10);
 );



functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end UserManager
```

11 UserTest

```
class UserTest is subclass of Data
types
-- TODO Define types here
values
-- TODO Define values here
instance variables
-- TODO Define instance variables here
 e : Event := new Event("evento1", 15, 10, mk_Date(10,1,2019));
 e2 : Event := new Event("evento2", 15, 10, mk_Date(29,2,2020));
 e3 : Event := new Event("evento3", 15, 10, mk_Date(29,4,2020));
 e4 : Event := new Event("evento4", 15, 10, mk_Date(28,2,2021));
```

```
 u : User := new User("dank", "memes");
 em : EventManager := new EventManager({ "evento1" |-> e, "evento2" |-> e2, "evento3" |-> e3, "
     evento4" |-> e4});
 um : UserManager := new UserManager({"dank" |-> u});
 tm : TicketManager := new TicketManager({|->});
 t : Ticket := new Ticket("dank", "evento1");
operations
-- TODO Define operations here

 private assertTrue: bool ==> ()
  assertTrue(cond) == return
  pre cond;


 private testGetName : () ==> ()
 testGetName() == (
  assertTrue(u.getName() = "dank");
 );


 private testAddFunds : () ==> ()
 testAddFunds() == (
  u.addFunds(100);
  assertTrue(u.getFunds() = 100);
 );


 private testRemoveFunds : () ==> ()
 testRemoveFunds() == (
  u.addFunds(100);
  u.removeFunds(10);
  assertTrue(u.getFunds() = 90);
 );


 private testGetPassword : () ==> ()
 testGetPassword() == (
  assertTrue(u.getPassword() = "memes");
 );


 private testAddEvent : () ==> ()
 testAddEvent() == (
  u.addEvent("eventTest1");
  u.addEvent("eventTest2");
  assertTrue(u.getEvents() = {"eventTest1","eventTest2"});
 );


 private testRemoveEvent : () ==> ()
 testRemoveEvent() == (
  u.addEvent("eventTest1");
  u.addEvent("eventTest2");
  u.removeEvent("eventTest1");
  assertTrue(u.getEvents() = {"eventTest2"});
 );


 private testAddTicket : () ==> ()
 testAddTicket() == (
  u.addTicket(0);
  u.addTicket(1);
  assertTrue(u.getTickets() = {0,1});
 );
```

```
private testRemoveTicket : () ==> ()
testRemoveTicket() == (
 u.addTicket(0);
 u.addTicket(1);
 u.removeTicket(0);
 assertTrue(u.getTickets() = {1});
);


private testBuyTicket : () ==> ()
testBuyTicket() == (
 u.addFunds(100);
 u.buyTicket(10,0);
 assertTrue(u.getTickets() = {0});
 assertTrue(u.getFunds() = 90);
);


private testLogout : () ==> ()
testLogout() == (
 assertTrue(um.login("dank", "memes"));
 assertTrue(um.getCurrentUser() = "dank");
 um.logout();
 assertTrue(um.getCurrentUser() = "");
);


private testWrongPass : () ==> ()
testWrongPass() == (
 um.register("testUser", "testPass");
 assertTrue(not um.login("testUser", "wrongpass"));
);


private testPromote : () ==> ()
testPromote() == (
 u.addFunds(100);
 assertTrue(um.login("dank", "memes"));
 um.promoteEvent("evento3", em);
 assertTrue(e3.getPopularity() = 10);
);


private testGetUser : () ==> ()
testGetUser() == (
 assertTrue(um.getUser("dank") = u);
);


private testGetUsers : () ==> ()
testGetUsers() == (
 assertTrue(um.getUsers()("dank") = u);
);


private testGetUserTickets : () ==> ()
testGetUserTickets() == (
 assertTrue(um.login("dank","memes"));
 u.addFunds(100);
 u.buyTicket(10,0);
 assertTrue(um.getUserTickets() = {0});
 assertTrue(u.getFunds() = 90);
);
```

```
 private testGetUserTicketsEvent : () ==> ()
 testGetUserTicketsEvent() == (
  assertTrue(um.login("dank","memes"));
  u.addFunds(100);
  um.buyTicket("evento4", tm, em);
  um.buyTicket("evento4", tm, em);
  um.buyTicket("evento4", tm, em);
  um.buyTicket("evento3", tm, em);
  assertTrue(um.getUserTicketsEvent("evento4", tm) = {33, 34, 35});
 );


 public static main: () ==> ()
 main() == (
  new UserTest().testGetName();
  new UserTest().testAddFunds();
  new UserTest().testRemoveFunds();
  new UserTest().testGetPassword();
  new UserTest().testAddEvent();
  new UserTest().testRemoveEvent();
  new UserTest().testAddTicket();
  new UserTest().testRemoveTicket();
  new UserTest().testBuyTicket();
  new UserTest().testLogout();
  new UserTest().testWrongPass();
  new UserTest().testPromote();
  new UserTest().testGetUser();
  new UserTest().testGetUsers();
  new UserTest().testGetUserTickets();
  new UserTest().testGetUserTicketsEvent();
 );
functions
-- TODO Define functiones here
traces
-- TODO Define Combinatorial Test Traces here
end UserTest
```