MFES

January 5, 2019

Contents

1 Event

```
class Event
types
-- TODO Define types here
  public String = seq of char;
  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
-- 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 : () ==> nat
 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 getEarnings : () ==> rat
 getEarnings() == (
  return card (tickets) * ticketPrice;
 )

functions
-- TODO Define functiones here
```

```
 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 Event
```

2 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 ==> nat
 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
```

3 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
```

4 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
```

5 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};



 public promoteEvent : () ==> ()
 promoteEvent() == (removeFunds(10))
 pre funds - 10 >= 0;

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

6 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 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).promoteEvent();
 );



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