13.2 Lotto example sketch

Description

The next example is an experiment on playing Lotto. In this simplified version of Lotto you have to guess seven different numbers in the interval from 1 to 34. You may submit one bet each week. At the end of the week the Lotto system chooses randomly seven different winner numbers, and the winning players are those that have submitted a bet with these winner numbers.

We use a `MonitorSystem` to represent the players and the Lotto. Lotto is represented by an object with `MonitorProcess` as superclass. A player is represented by an instance of class `Player` that is a subclass of `MonitorProcess`.

The `Lotto` system has the following overall structure:

```
LottoExperiment: obj MonitorSystem
   mnoOfPlayers: val 500
   betSize: val 7
   class Hand:
      :::
   class Bet(thePlayer: ref Player, theHand: ref Hand):
      :::
   class Player(inx: var integer): MonitorProcess
      :::
   Lotto: obj MonitorProcess
      :::
   Lotto.start
   generatePlayers: do
      ...
```

`LottoExperiment` has the following main attributes:

- Constants `noOfPlayers` and `betSize`, which defines the number of `Player` objects and the size of a `Hand` in a `Bet`.
- Class `Hand` which represents a hand of seven numbers.
- Class `Bet` which represents a bet. It has parameters `thePlayer`, which is the `Player` that has submitted the `Bet` and `theHand`, which is the `Hand` of the `Bet`.
- Class `Player` which represents a player.
- An object `Lotto`, which represents the Lotto.
- A statement that starts the `Lotto` process.
- An object `generatePlayers` which is an object that generates the players.

Class `Player` has the following structure:

```
   class Player: MonitorProcess
      cycle
         Lotto.submit(Bet(this(Player),Hand)
         doSomethingElse
```

- A `Player` repeatedly submits a `Bet` by invoking `Lotto.submit`.
- The argument of `submit` is the object `Bet(this(Player), Hand)`.
- The first argument of `Bet` is a reference to the `Player` submitting the `Bet`.
- The second argument of `Bet` is a `Hand` which randomly generates 7 numbers.
- After submitting a `Bet`, it executes `doSomethingElse` before playing the next time; `doSomethingElse` is not specified here.

The Lotto is represented by an object `Lotto`. This object keeps the bets being submitted, it has the winning bet, and the deadline for submitting bets. As players in parallel submit bets by calling the method `submit`, the `Lotto` object is defined

as a `MonitorProcess` with `submit` as an entry method — similar to en `entry`-method of a `Monitor` object.

The structure of `Lotto` is as follows:

```
   Lotto: obj MonitorProcess
      submit(B: ref Bet): entry
         bets.insert(B)
      clearBets:
         bets.clear
      findWinningBets:
         :::
      bets: obj Set(#Bet)
      :::
```

`Lotto` has the following attributes:

- An `entry`-method `submit`, which may be used by `Player` objects to `submit` bets. As mentioned, it works in the same way as an `entry` method of a `Monitor`. This means that at most one `submit` method may be executed at a given time.
- The `Bets` being submitted are stored in `bets` which is a `Set`.
- Methods `clearBets` and `findWinningBets` that are private to `Lotto` in the sense that they may not be invoked by objects (here `Players`) outside `Lotto`. As for `Monitor`, only methods being submethods of `entry` may be invoked from outside `Lotto`.

As said, a Lotto-period is a week. During a week the `Players` may submit bets. At the end of a week, Lotto stops accepting `Bets` via `submit`. It then invokes `findWinningBets` to find the possible winners of the week.

When possible winners have been found, `Lotto` informs the winners. It invokes `clearBets` to remove all elements from `bets` and open up for a new round of submissions. This is represented by the following statements of `Lotto`:

```
Lotto: obj MonitorProcess
   -"-
  run: do
      waitAndAccept(aWeek)
      findWinningBets
      clearBets
      restart(run)
```

The object `run` is executed forever:

- When a `MonitorProcess` (here `Lotto`) is executing, it is not possible to execute an `entry`-method from outside.
- The method `waitAndAccept` is defined as an attribute of `MonitorProcess`.
- When `waitAndAccept(aWeek)`, the `MonitorProcess` (`Lotto`) waits for a period – here defined by `aWeek` and while waiting it accepts `entry`-methods like `submit`.
- When `Lotto` has waited for a week, `waitAndAccept` resumes execution and closes for execution of `entry`-methods like `submit`.
- It then executes `findWinningBets` followed by `clearBets`.

When `Lotto` executes `waitAndAccept`, there may be `Players` that have invoked `Lotto.submit`. These invocations may be executed before `waitAndAccept` closes for execution of `entry`-methods. This means that a `Player` may submit a `Bet` after a week has passed. From a modeling point-of-view this may be ok since the invocation of `submit` has happened before a week has passed. If one does not consider this to be ok, the implementation of `waitAndSubmit` has to be able to block `Players` waiting to execute a `submit`.

We now show the details of `findWinningBets`:

```
   findWinningBets:
      winningBet := Bet(Player("Winner"), Hand)
      bets.scan
          if (current.equal(winningBet)) :then
             "Winner: ".print
             current.print
      newline
```

- As mentioned, it starts by assigning `possibleToSubmit` to false.
- It then generates a `winningBet` with a hypothetical `Player` called “Winner” and a `Hand` with seven random numbers as arguments
- It then scans the submitted bets in `bets` and checks whether or not the current `Bet` is equal to the `winningBet`
- If a winner is found, the `Player` of this `Bet` is printed.

Finally we show the structure of class `Hand` and class `Bet`:

```
class Hand:
   numbers: obj Array(betSize,#integer)
   for (1):to(betSize):repeat
        numbers.put(random(1,34)):at[inx]

class Bet(thePlayer: ref Player, theHand: ref Hand):
   equal(aBet ref Bet) -> B: var boolean:
      ...
```

- Class `Hand` has an array `numbers` that contains the seven random numbers of the `Hand`.
- It uses a `for:to:repeat` to generate and store seven random numbers in `numbers`.
- Class `Bet` has an `equal` method in addition to its two parameters.

For completeness, all the code for the Lotto example is shown below:

```
LottoExperiment: obj MonitorSystem
   mnoOfPlayers: val 500
   betSize: val 7
   class Hand:
      numbers: obj Array(betSize,#integer)
      for (1):to(betSize):repeat
           numbers.put(random(1,34)):at[inx]
   class Bet(thePlayer: ref Player, theHand: ref Hand):
      equal(aBet ref Bet) -> B: var boolean:
         ...
   class Player(inx: var integer): MonitorProcess
       cycle
          Lotto.submit(Bet(this(Player),Hand)
          doSomethingElse
   Lotto: obj MonitorProcess
      submit(B: ref Bet): entry
         bets.insert(B)
      clearBets:
         bets.clear
      findWinningBets:
         winningBet := Bet(Player("Winner"), Hand)
            bets.scan
              if (current.equal(winningBet)) :then
                 "Winner: ".print
                 current.print
              newline
      bets: obj Set(#Bet)
      run: do
         waitAndAccept(aWeek)
         findWinningBets
         clearBets
         restart(run)
   Lotto.start
   generatePlayers: do
      ...
```