Timeclock Web Services Interface (Ruby Version)

Session

All communication with the server is through a Session object. To obtain one, do this:

```
require 'timeclock-web-services'
session = start_session()
```

Once that's done, communicate with the server by sending messages to the session. The following list is a subset sufficient for this course. Each is described more fully later.

In the list below, each command is summarized with an example of typical use. Return values are assigned to variables whenever the return value is meaningful. Some methods can raise TimeclockError instead of returning. That's not shown here, but is described later.

```
session.accept_job(job)

hash_with_job_names_as_keys = session.jobs


job = session.start(job_name_string, desired_time)

true_or_false = session.running?(job_name_string)


job = session.pause(desired_time)

true_or_false = session.paused?(job_name_string)


record = session.stop(job_name_string, desired_time)

array_of_records = session.stop_all(desired_time)

true_or_false = session.stopped?(job_name_string)


array_of_records = session.records


session.forget_everything
```

Objects returned by session methods

Job

A job is something you can start, pause, or stop. Before you can start a job, you have to give it to the session with `session.accept_job`.

```
job = Job.named('a name string')
```

This creation method takes a String name and returns a Job with that name.

```
String = job.name
```

Any Job responds to `name` with the name string used to create it.

```
job1 == job2
```

Two jobs are considered equal if their names are the same. So you can do either this:

```
assert_equal(job1, job2)
```

Or this:

```
assert_equal(job1.name == job2.name)
```

Record

Whenever a job is started, a Record is created. It accumulates time whenever it's running (isn't currently paused and has never been stopped). Once the job is stopped, the record is finished and doesn't change thereafter. If the job is started again, a new record is created.

```
true_or_false = record.running?
true_or_false = record.paused?
true_or_false = record.stopped?
```

These three methods are used to tell if a record is running, paused, or stopped.

```
time = record.time_started
```

This returns a Time object. (See *Programming Ruby* for much detail on it. In this course, we'll probably ignore those details and not test whether a record was created at the right time.)

```
job = record.job
```

This returns the job that was started to create this record.

```
seconds = record.time_accumulated
```

This returns the number of seconds the record has accumulated. If a Record is running, the `time_accumulated` is updated every time you ask for it, so two calls may produce different values.

Session messages

All of these messages are sent to Session objects.

accept_job(job)

`job` must be a Job object. The session now knows that `job` exists. The Job can now be started. It will be returned by `jobs`.

Example:

```
job = Job.named('hello')
session.accept_job(job)
assert_equal(true,
             session.jobs.has_key?('hello'))
```

forget_everything

The session forgets all the jobs recorded with `accept_job` and all the records created by starting jobs. `jobs` will now return an empty Hash. `records` will now return an empty array.

Example:

```
session.forget_everything
assert_equal({}, session.jobs)
assert_equal([], session.records)
```

jobs

Returns a Hash containing all the jobs accepted with `accept_job`. The keys are strings that name jobs. The values are the jobs themselves.

Example:

```
jobs = session.jobs
assert_equal('hello', jobs['hello'].name)
```

pause(desired_time)

Pauses the currently running job as of the `desired_time`, which must be a Time object. A paused job accumulates no time until it's resumed. Returns the Job object paused.

A TimeclockError is raised if no job is running.

Example:

```
paused_job = session.pause(Time.now)
assert_equal(true, session.paused?(paused_job.name))
```

paused?(job_name_string)

Returns true if the job named by the `job_name_string` is paused, false if it is running or stopped.

Raises a TimeclockError if `job_name_string` names no job the Session knows about.

Example:

```
assert_equal(true, session.paused?('hello'))
```

records

Returns an Array of all the records the Session knows about. Each entry was created when a job is started with `start`. Within the array, the records are in the order they were started. (The first-started record is first, the next-started is next, and so on.)

The array contains both "finished" records (those whose jobs have been stopped) and those that can still accumulate more time (are running or paused).

Example:

```
records = session.records
assert_equal('hello', records[0].job.name)
```

running?(job_name_string)

Returns true if the job named by the `job_name_string` is running, false if it is paused or stopped.

Raises a TimeclockError if `job_name_string` names no job the Session knows about.

Example:

```
assert_equal(true, session.running?('hello'))
```

start(job_name_string, desired_time)

Starts the job named by `job_name_string` as of `desired_time`, which must be a Time object. There are two cases:

- The job was stopped. Time begins accumulating. The job will be running until it is paused or stopped.
- The job was paused. It is resumed. Time continues accumulating.

The started job is returned.

If any other job was running when the `start` message was sent, it is paused.

There are two error cases that cause `start` to raise a TimeclockError.

- The `job_name_string` names no job the session has accepted (with `accept_job`).
- The job is already running.

Example:

```
session.start('hello', Time.now)
assert_equal(true, session.running?('hello'))
```

stop(job_name_string, desired_time)

Stops the job named by `job_name_string` as of `desired_time`, which must be a Time object. No more time can be accumulated in the job's current record. If the job is started again, a new record is created.

The "finished" record is returned.

There are two error cases that cause `start` to raise a TimeclockError.

- The `job_name_string` names no job the session has accepted (with `accept_job`).
- The job is neither running nor paused.

Example:

```
start_time = Time.now
session.start('hello', start_time)
record = session.stop('hello', start_time + 1.minute)
assert_equal(1.minute, record.time_accumulated)
```

stop_all(desired_time)

Stops all running or paused jobs as of `desired_time`, which (surprise!) must be a Time object.

The return value is an array of the resulting "finished" records. They are in order. (The first one started is the first element of the array, etc.)

Think of this method as calling stop on each job, packaging up the results in an array, and returning that.

Example:

```
first_start_time = Time.now
session.start('first', first_start_time)

second_start_time = first_start_time + 1.hour
session.start('second', second_start_time)

stopped = session.stop_all(first_start_time + 3.hours)

assert_equal(first_start_time,
             stopped[0].time_started)

assert_equal(second_start_time,
             stopped[1].time_started)
```

stopped?(job_name_string)

Returns true if the job named by the `job_name_string` is stopped, false if it is paused or running. A stopped job may be one that's been accepted (with `accept_job`) and never started, or one that's been started and subsequently stopped.

Raises a TimeclockError if `job_name_string` names no job the Session knows about.

Example:

```
session.accept_job(Job.named('hello'))
assert_equal(true, session.stopped?('hello'))

session.start('hello', Time.now)
assert_equal(false, session.stopped?('hello'))

session.stop('hello', Time.now)
assert_eqal(true, session.stopped?('hello'))
```