CS 383- 60%
Project ZERO
Kodi Cumbo
Contributor KC, GitHub Quiksilv26

Testing plan

Catch uses a SECTION and REQUIRE format for test cases.
For each SECTION in a TEST_CASE, the entire TEST_CASE is executed from the start. SECTIONS can be nested. For each nested SECTION, the enclosing SECTION is executed from the start. Each leaf-SECTION (that contains no other SECTIONS nested within) is executed exactly once.
Example:

- TEST_CASE
  - Set-up A
  - REQUIRE i
  - SECTION 1
    - REQUIRE ii
    - REQUIRE iii
  - SECTION 2
    - Set-up B
    - REQUIRE iv
  - SECTION 3
    - Set-up C
    - REQUIRE v
  - SECTION 4
    - Set-up D
    - SECTION 5
      - Set-up E
      - REQUIRE vi
    - SECTION 6
      - Set-up F
      - REQUIRE vii

In this example, SECTIONs 1,2, and 3 are run with Set-up A. SECTION 3 is run with Set-up A and C. SECTION 6 is run with Set-up A, D, and F.

Catch has handy Macros SCENARIO, GIVEN, WHEN, THEN, which map to TEST_CASE and SECTION and prettify the output a bit. They will be used with the format above to enumerate my testing plans on the following pages.

Additional notes: TEST_CASEs and SCENARIOs can be called individually as command line arguments, or by description, or by tag.

The “make test” and “make KCTest” compiled master files retain all sub-divisions and the run_tests and run_KCtests executables they generates can be used to run them.

ATTRACTUION RANKERS:

SCENARIO- “Stub lists are created appropriately and rating and sorting work”

- Set-up: KCAttractionRankerMaster masterList
- GIVEN: “A RankerMaster fed a list of mixed Attraction types, with known attributes”
  - Set-up: Array of formatted attractions (or struct/macro dummies) with known attributes- attrList
  - Set-up: masterList.buildStubList(attrList)
  - WHEN: “Stub list has just been created”
    - THEN: “mCheapest has been set correctly”
      - REQUIRE: mCheapest is equal to expected
    - THEN: “Appropriate Attractions have been added to list, with rating initialized to zero
      - REQUIRE masterList.getTopAttraction == &attrList[0]
      - REQUIRE masterList.getRankedAttraction(1) == &attrList[1]
      - And so on. masterList does not include any Coin stands
  - WHEN: “Attractions are rated and sorted”
    - THEN: “They are in descending order by rating”
      - REQUIRE sorted mStubList
    - THEN: “They match the expected order
      - REQUIRE expected order
  - WHEN: “An attractions attributes are altered”
    - REQUIRE: highest rated attraction has appropriate rating
    - Set-up: Change highest rated attraction attributes
    - Set-up: RateAttraction called on former highest rated attraction
    - THEN: “RateAttraction changes the rating appropriately”
      - REQUIRE: new rating matches expectation
    - THEN: “SortStub places the newly rated attraction in the right spot”
      - REQUIRE: New order matches expectation
- GIVEN: “A RankerMaster fed a list with games of various price and several identical rides with different prices
  - Set-up: attrGamePriceCheck created appropriately
  - Set-up: Stub list built
  - WHEN: “Stub List has been rated and sorted”
    - Set-up: Rate and sort
    - THEN: “The exchange rate of coins to money is correct”
      - REQUIRE: correct order of games and rides
- GIVEN: “A RankerMaster fed a list that includes games but no coin stands”
  - Set-up: appropriate list, stub list built
  - WHEN: “Stub List has been rated and sorted”
    - Set-up: Rate and sort
    - THEN: “Games have lowest possible rating”
      - REQUIRE: correct order of attractions

- REQUIRE: all games have appropriate rating

— GIVEN: "A RideRanker fed a list of attractions of mixed attributes"
  - Set-up: Appropriate list created
  - Set-up: masterList.mRideList created and fed attraction list
  - WHEN: "Stub List has just been created"
    - ¬ THEN: "Only rides are in list"
      - REQUIRE: only rides in stub list
    - ¬ THEN: "mCheapest, mLeastThrilling, mLeastNauseating set correctly"
      - REQUIRE: members match expected values
  - WHEN: "Stub List has been rated and sorted"
    - ¬ Set-up: Rate and sort
    - ¬ THEN: "Ride order matches expected"
      - REQUIRE: ride order is as expected

— GIVEN: "A VendorRanker fed a list of attraction of mixed attributes"
  - Set-up: Appropriate list created
  - Set-up masterList.mVendorList created and fed attraction list
  - WHEN" Stub list has just been created"
    - ¬ THEN: "Only vendors are in list"
      - REQUIRE: only vendors in stub list
    - ¬ THEN: "mCheapest set correctly"
      - REQUIRE: mCheapest matches cheapest food item offered
  - WHEN: "Stub list has been rated and sorted"
    - ¬ Set-up: rate and sort
    - ¬ THEN: "Vendor order matches expected"
      - REQUIRE: vendor order is as expected

— GIVEN: "A GameRanker fed a list of attractions of mixed attributes"
  - Set-up: Appropriate list created
  - Set-up masterList.mGameList created and fed attraction list
  - WHEN" Stub list has just been created"
    - ¬ THEN: "Only Games are in list"
      - REQUIRE: only Games in stub list
    - ¬ THEN: "mCheapest set correctly"
      - REQUIRE: mCheapest matches cheapest coin-price
  - WHEN: "Stub list has been rated and sorted"
    - ¬ Set-up: rate and sort
    - ¬ THEN: "Game order matches expected"
      - REQUIRE: Game order is as expected
  - GIVEN: "A CoinStandRanker is fed a list of mixed attractions"
    - ¬ Set-up: masterList.mGameList->mCoinStandList created and fed attraction list
      - WHEN: "Stub list has just been created"
        - THEN: "Only Coin Stands are in list"
          - REQUIRE: Only stands in list
        - THEN: "mCheapest and mSmallest set correctly"
          - REQUIRE: members have expected values

- WHEN: “Stub list has been rated and sorted”
    - THEN: “Stand order matches expected”
        - REQUIRE: Stand order as expected

CHOOSER:

SCENARIO: “Chooser processes customers as expected”

- Set-up: KCChooser decisionMaker
- GIVEN: “A masterList with sorted main and sub-lists of mixed attribute attractions”
    - Set-up: Create masterList and sub-lists. Feed, rate, and sort attraction list of mixed attribute and status
    - GIVEN: “An average customer set as subject”
        - Set-up: Create average customer object
        - Set-up: Set as subject
        - WHEN: “Subject is at an attraction”
            - Set-up: Busy flag set
            - THEN: “statusCheck returns busy”
                - REQUIRE: statusCheck returns busy
        - WHEN: “Subject is in a queue”
            - Set-up: Queue flag set
            - THEN: “statusCheck returns queued”
            - WHEN: “Customer is happy”
                - THEN: “patienceCheck decides to stay, no stat penalty”
                    - REQUIRE: patienceCheck returns stay
            - WHEN: “Customer is hungry”
                - Set-up: change hunger
                - THEN: “patienceCheck decides to stay, requests stat penalty”
                    - REQUIRE: patienceCheck returns get_upset
            - WHEN: “Customer has been in the queue for several ticks”
                - Set-up: customer has been enqueued for some time
                - THEN: “patienceCheck decides to stay, requests penalty”
                    - REQUIRE: patienceCheck returns get_upset
            - WHEN: “Customer is extremely hungry”
                - Set-up: customer set to very hungry
                - THEN: “patienceCheck decides to leave”
                    - REQUIRE: patiencCheck returns leave
            - WHEN: “Customer has been in the queue for very many ticks”
                - Set-up: customer enqueued for excessive time
                - THEN: “patienceCheck decides to leave”
                    - REQUIRE: patienceCheck returns leave
            - WHEN: “Customer is extremely nauseas”
                - Set-up: customer set to high nausea
                - THEN: “customer decides to leave”

        - REQUIRE: patienceCheck returns leave
- WHEN: Subject is free
    - THEN: “Customer picks top attraction”
        - REQUIRE: attractionSelect returns masterList.getTopAttraction
    - WHEN: “The top attraction is too expensive”
        - Set-up: top rated attraction has cost increased (is not re-rated or re-sorted)
        - THEN: “Customer picks second best”
            - REQUIRE: attractionSelect returns the right attraction
        - WHEN: “The top 5 attractions cost too much”
            - Set-up: top rated attractions have cost increased
            - THEN: “Customer picks 6th best”
                - REQUIRE: attractionSelect returns the right attraction
        - WHEN: “The customer has no money or coins”
            - Set-up: customer wallet stolen
            - THEN: “Customer leaves”
                - REQUIRE: attractionSelect returns NULL and leave command
            - WHEN: “The customer has no money, but has coins”
                - Set-up: customer steals a coin-purse
                - THEN: “Customer picks top game”
                    - REQUIRE: attractionSelect returns appropriate selection
    - WHEN: “The top attraction is too thrilling”
        - Set-up: top rated attraction has thrill-rating increased
        - THEN: “Customer picks second best”
            - REQUIRE: attractionSelect returns appropriately
    - WHEN: “The top attraction is too nauseating”
        - Set-up: Customer nausea increased
        - Then: “Customer picks second best”
            - REQUIRE: attractionSelect returns appropriately
    - WHEN: “The top attraction is a game”
        - Set-up: game in master list has satisfaction increased drastically, is re-rated and re-sorted
        - WHEN: “Customer has enough coins”

- THEN: “attractionSelect returns top game”
    - REQUIRE: attractionSelect returns appropriately
- WHEN: “Customer does not have enough coins”
    - Set-up: customer coins removed
    - THEN: “attractionSelect returns top coin stand”
        - REQUIRE: attractionSelect returns a coin stand
    - WHEN: “Coin stand queues adjusted”
        - Set-up: previous top coin stand has queue length increased, re-rated and re-sorted
        - THEN: “A different stand is selected”
        - REQUIRE: attractionSelect returns the appropriate stand

- WHEN: “The customer is very hungry”
    - Set-up: Customer hunger adjusted
    - THEN: “Customer picks top food stand”
        - REQUIRE: customer goes to food stand
    - WHEN: “The customer is also very nauseas”
        - Set-up: Customer nausea set
            - REQUIRE: customer goes to game
        - WHEN: “Customer has no coins”
            - Set-up: Coin purse stolen
                - REQUIRE: customer goes to coin stand
            - WHEN: “Customer has money enough for rides or food, but insufficient money to purchase coins
                - Set-up: customer nearly broke
                - REQUIRE: attractionSelect returns leave command
- WHEN: “The customer has a very large number of coins
    - Set-up: Customer finds a large bag of coins
        - THEN: ”Customer picks top game”
            - REQUIRE: customer picks top game

CROWD DIRECTOR:

SCENARIO: “Crowd Director builds ranker lists and gets population, updates crowd appropriately

- Set-up: KCCrowdDirector bossCD
- Set-up: Attraction list and population created with mixed but known attributes
- Set-up: bossCD.getRides and bossCD.getPopulation run
- GIVEN: “A set up crowd director”
  - WHEN: “Crowd director is run once”
    - Set-up: run updateCrowd
    - THEN: “All customers queued”
      - REQUIRE: All customers in population have queue flag set
    - THEN: “Customers chose different rides”
      - REQUIRE: queue-length of attractions checked, more than just top attraction got selected
    - WHEN: “Crowd director is run again”
      - Set-up: run updateCrowd
        - THEN: “Attraction ranking order has changed”
          - REQUIRE: mMasterList order has changed from original, due to queue lengths
      - WHEN: “Crowd director is run repeatedly”
        - Set-up: repeat updateCrowd several times
        - THEN: “Crowd shows multiple statuses and has changed attributes”
          - REQUIRE: some customers have finished attractions and have altered stats, some are on attractions, some are in queues