

Testing 2, Project 1

Implement a database for tracking Magic: The Gathering (MTG) cards, using TDD.

Due: 11:59PM, Tuesday 10/4/2004

Building on the code you or your partner wrote for HW #1, implement the following user stories in Java, using test-driven development:

Story 1. Card Type

Each card has a card type. In MTG, the card types are: Creature, Land, Sorcery, Enchantment, Instant, Artifact, and Artifact Land. When entering a card, the user should be able to choose the type from a list rather than having to type it.. (Note: A card that says “Summon XXX” in the type line is considered a Creature card.)

Task 4-1. Add support for a card to have a single card type

Task 4-2. Require a card type for each card

Task 4-3. Show the card type in the GUI next to the card name.

Task 4-4. Allow the card type to be edited

Story 2. Casting Cost

Cards can have a casting cost. Costs are indicated by using numbers for colorless mana, and letters for colored mana (B =- black, U = blue, G = green, R = red, W = white). For example, if a card cost 2 colorless (or any color) mana and 2 black mana, its cost would be written 2BB. If a card cost a green and a white mana, its cost would be GW. It is possible for cards to not have a casting cost (e.g., lands have no casting cost associated with them). A casting cost of 0 is different than no casting cost. The GUI should display a card’s casting cost.

Story 3. Color

Each card has at least one color associated with it. A card’s colors are determined by the colors of mana required to cast the spell. Possible colors are Black, Blue, Green, Red, White, Colorless, and Land. Note that Colorless only applies when there is no colored mana in the cost and all lands are of color land, regardless of what color mana they may produce. Colors should be determined for a card by analyzing the card’s casting cost. The GUI should indicate a card’s color.

Story 4. Rarity and Set

Each card belongs to at least one set. In a given set, that card has a rarity associated with it. The possible rarities are “Common”, “Uncommon” and “Rare”. The sets can be found at <http://www.wizards.com/default.asp?x=magic/rules/faqs>. It is possible for one card to be in multiple sets with different rarities in each set. The GUI should display all the sets and rarities for a selected card.

Story 5. Power and Toughness

All cards that are of type “Creature” or “Artifact Creature” have a power and toughness associated with them to indicate how much damage they give and how much they can take. Non-creature cards have no such value. The user should be required to enter this value for creatures but not required for other types of cards. The GUI should display this information when it is applicable to the currently displayed card. Some creatures specify their power and/or toughness in ways such that the number can only be determined when the card is in play. These cards have the non-determinable values displayed as *.

Story 6. Rules Text

Some cards have additional abilities that are listed below the artwork. The database should track these abilities. Flavor text and reminder text (in other words, all the italicized text) does not need to be tracked. Costs for abilities are to be entered in the same form as casting cost, with the addition of using T to represent tapping the card. For example, the card has an activated ability where its controller can pay 1 mana (of any color), tap the card itself, and sacrifice the card to add one mana of any color to her mana pool and draw a card. This ability would be entered: “1,T, Sacrifice Chromatic Sphere: Add one mana of any color to your mana pool. Draw a card.” The GUI should display this text for the active card.

Story 7. Persistence

Card data is persistent across application sessions. The GUI should be able to save a collection of cards off to a new file, save to the same file as was last saved, and load a selection of cards from a file.

Story 8. Quantity

Each card in the database should have a quantity associated with it that represents the number of that card the user owns. This should be editable and displayed on the GUI. A card should have a quantity value for each set it's in.

Story 9. Cards can be combined into decks

Magic: The Gathering games are played with decks of magic cards. A card can belong to multiple decks. Enable the user to define and store these decks in the database. The GUI should provide a way to view the decks, view the number of cards in a deck, add or remove cards from decks, and display what decks a selected card is in.

Story 10. Decks can have more than one of a given card

Decks are allowed to have up to 4 instances of a particular card (regardless of set) for any card except basic lands (Forest, Island, Plains, Mountain, and Swamp). (You do not have to worry about the card “Relentless Rats” which also exempts itself from this restriction.) The GUI should allow people to add more than one of a card, even if the database indicates they have fewer cards than they wish to add.

Story 11. Decks are persistent

Once a user has created a deck, they should be able to save the deck and reload it later.

Story 12. Cards in the database can be filtered

Once there are cards in the database, the total list can be filtered so that only a subset of cards is visible. This filtering can be done on any attribute of a card. You don't have to worry about filtering on multiple attributes at one time.

Submission Guidelines

Submit your final project code through the subversion server. This archive should contain:

- All code files in their final state. The code should be compilable and run as specified in the user stories. All the tests should pass.
- A document describing the process you went through to develop the application. This document should include enough iterations (at least 3) to demonstrate that you understand (and followed) the concepts of TDD, but does not need to include every single step you take.
- A document containing a project retrospective, done after you have completed the rest of the project. This retrospective should include discussion about the project. Be sure to answer the following questions in your discussion:
 - Does the final project code look different than you originally expected it to or what you would normally expect had you developed it without following TDD?
 - What things surprised you most about the process of developing the application?
 - Are there things you would do differently in retrospect, if you had the project to do over again?
 - Was there a time difference in the length of time it took to develop the application using TDD than what you would expect if you had developed it the way you have for past classes?
 - Did you notice any differences that arose because of having a pair of people working on developing the application?

Remember to put your name in all of the files you submit, and be specific in all your answers.

Notes

- While much of this project parallels the example in Astels' book, you are not constrained to follow the same path he did, provided you adhere to the guidelines of TDD and programming by intention. If you have strong disagreements with his approach, document the disagreements in your process documentation. Both members of the pair should be prepared to discuss these disagreements if called upon.
- Your code should be refactored as much as possible.
- You are strongly encouraged to work on this project in pairs, with each pair working together on one computer. Choosing to work solo will not result in any relaxation of work expectations or grading.