The Zet Game

The SET Game

SET is a simple card game which is gaining popularity.

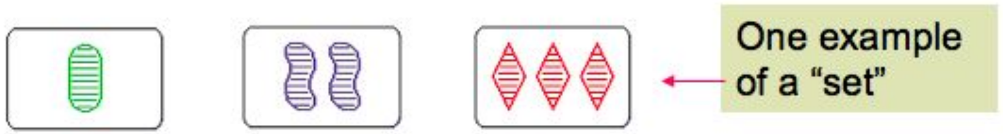
http://www.setgame.com has the official rules, examples, game variations, and other resources.

SET® is a registered trademark of SET Enterprises, Inc.

Since it is a registered trademark, we have decided to call our game Zet!

The ZET Game (cont'd)

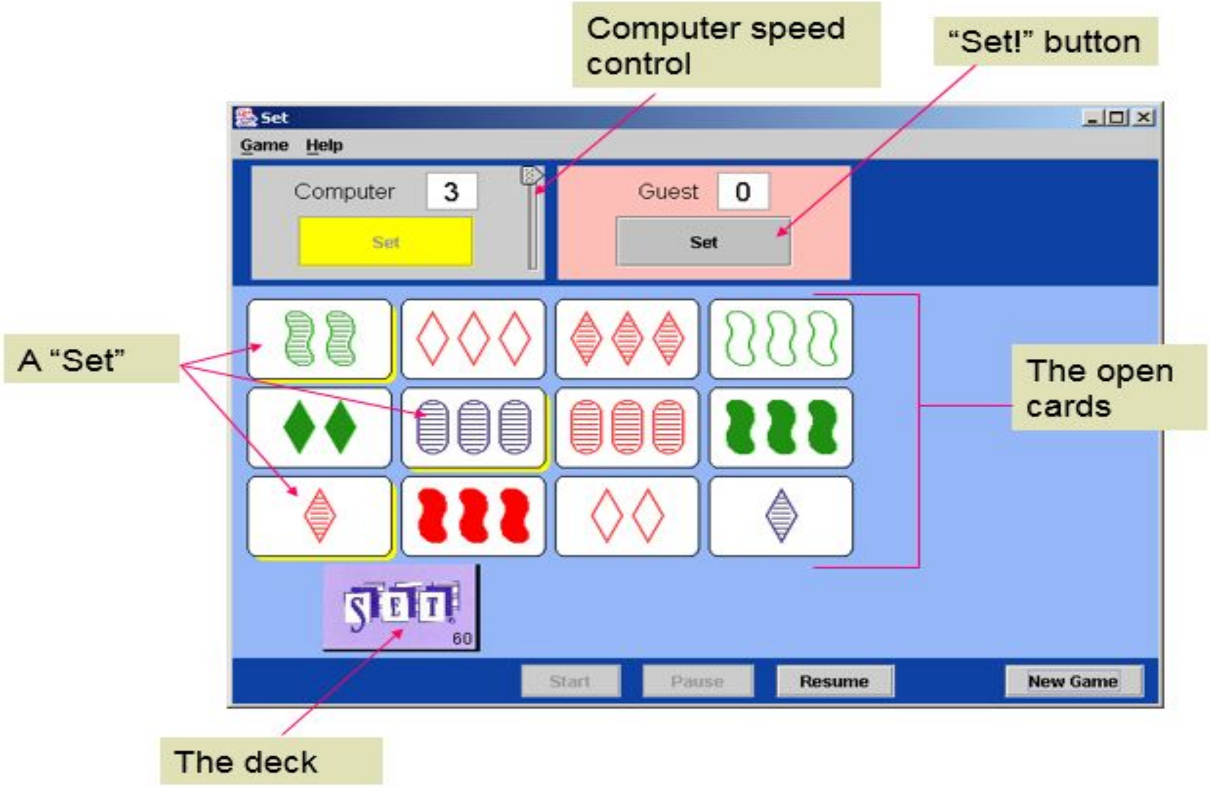
- A ZET deck consists of 81 cards. A card has four attributes:
  - number of symbols (1, 2, or 3)
  - symbol shape (oval, squiggle, or diamond)
  - symbol fill (outlined, striped, or solid)
  - symbol color (red, green, or blue)
- A "zet" is three cards, such that for each attribute its values for the cards are either **all** the same or **all** different.



The SET Game (cont’d)

- At the start of the game, 12 cards are open and all the players look for a “set” in them.
- The player who sees a set announces, “Set!” then promptly points to the three cards of the set.
- If the cards indeed form a set, these cards are removed and replaced with three cards from the deck, and the player gets one point; otherwise, the cards remain on the table, and the player loses one point.
- If all the players agree that the open cards don’t have any sets, 3 additional cards are opened.

Computer ZET

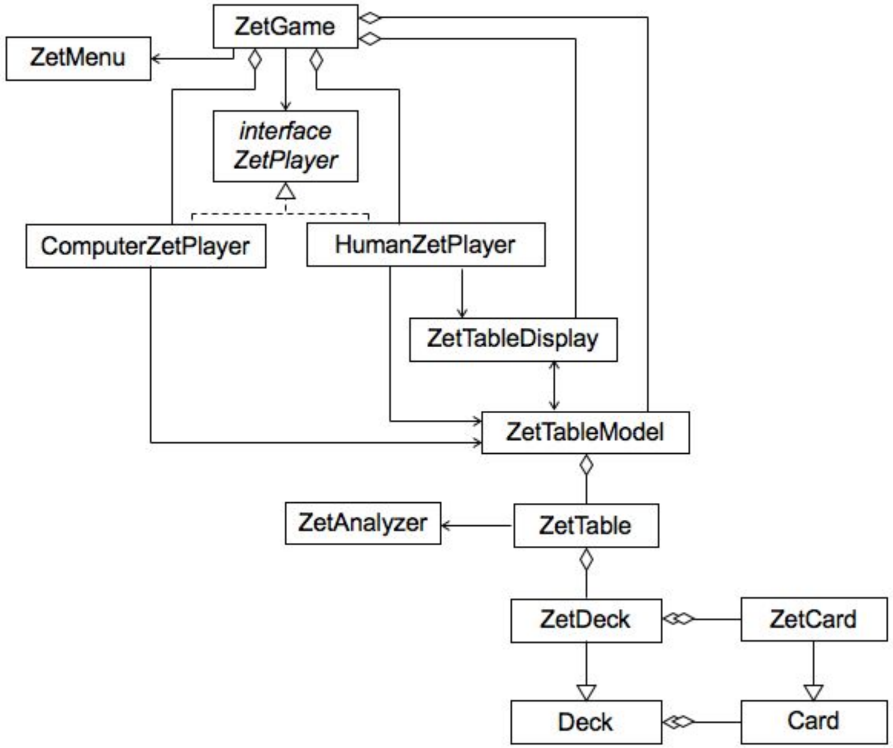


Demonstration!!!

Data Structures and Requirements

- We used networking in creating our game, and this included multiple data structures
    - A queue is used for storing incoming messages in order of arrival in the Hub
    - A TreeMap<Integer, ConnectiontoClient> is used by another thread in the hub to associate the ID integers of connected clients to the objects that handle their connections

Classes



- **ComputerZetPlayer:** Displays computer's control panel, handles timer events, keeps computer's score
- **ZetGameModel:** Keeps track of the card table and the picked cards, updates the display as necessary
- **HumanZetPlayer**: Displays guest's control panel, handles keyboard / mouse, keeps guest's score
- **ZetTableDisplay**: Draws the open cards, picked cards, and the deck
- **ZetGame**: Consolidates GUI, creates and manages the players and the game model

Our Classes

- **Card**: represents a generic Card with a given int ID
- **ZetCard**: represents a card for the game of Set with four attributes: number: (1, 2, or 3), shape: (oval, squiggle, or diamond), fill (outlined, solid, or striped), and color (red, green, or blue).
- **Deck**: Represents a generic deck of cards
- **ZetDeck**: Holds 81 ZetCards
- **ZetAnalyzer**: Provides static methods for finding sets.
- **ZetTable**: Represents a card table with a deck and an array of open cards for the game of Set.

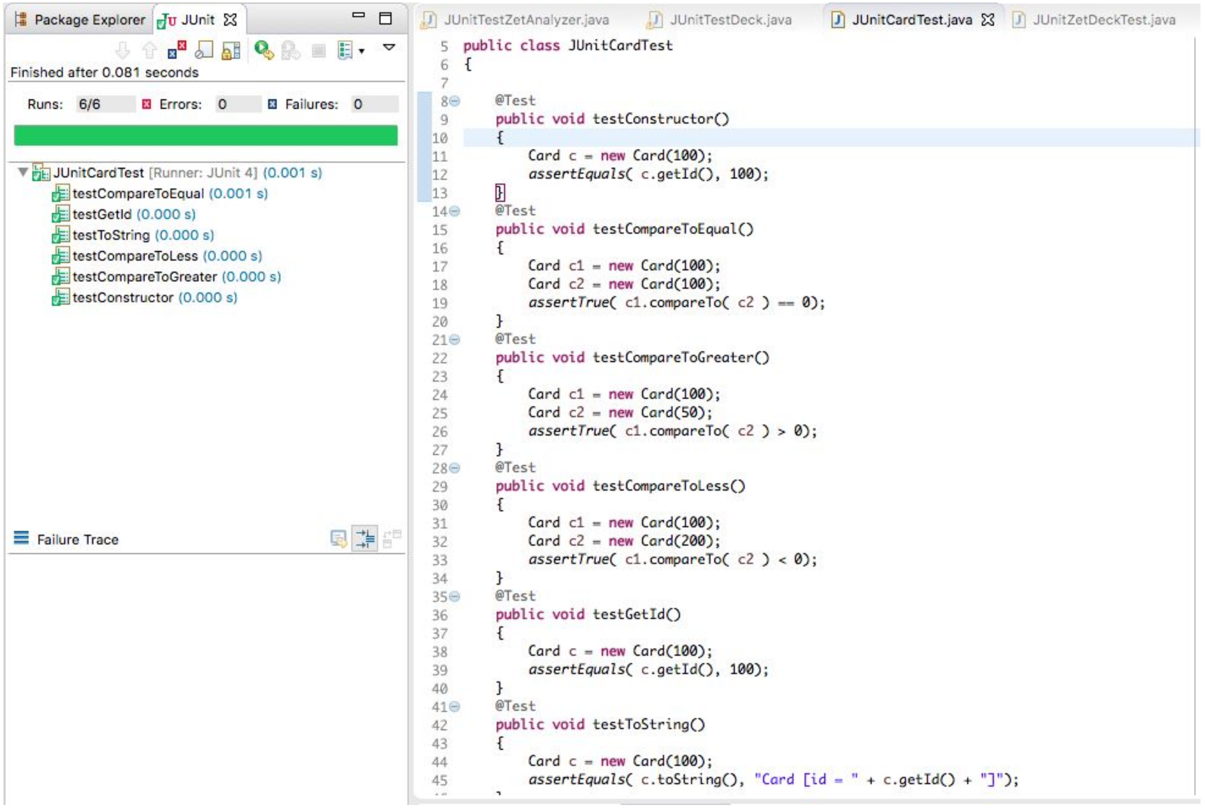
Development

We started with the more abstract classes like Deck and Card. We then moved onto ZetDeck and ZetCard, which extended the initial classes. Finally we worked on ZetAnalyzer and ZetTable because these classes performed operations on instances of the other classes.

Problems that we encountered:

- Constructing and displaying the cards with the GUI function
- Constructing cards in Deck vs ZetDeck
- The pause and resume functionality of our game

Testing



Package Explorer | JUnit

Finished after 0.081 seconds

Runs: 6/6 Errors: 0 Failures: 0

```
JUnitCardTest [Runner: JUnit 4] (0.001 s)
    testCompareToEqual (0.001 s)
    testGetId (0.000 s)
    testToString (0.000 s)
    testCompareToLess (0.000 s)
    testCompareToGreater (0.000 s)
    testConstructor (0.000 s)
```

Failure Trace

JUnitTestZetAnalyzer.java | JUnitTestDeck.java | JUnitCardTest.java | JUnitZetDeckTest.java

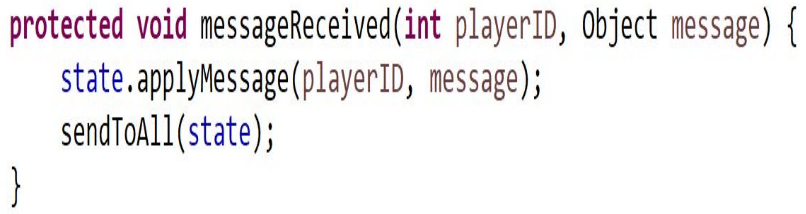
```java
public class JUnitCardTest
{

    @Test
    public void testConstructor()
    {
        Card c = new Card(100);
        assertEquals( c.getId(), 100);
    }
    @Test
    public void testCompareToEqual()
    {
        Card c1 = new Card(100);
        Card c2 = new Card(100);
        assertTrue( c1.compareTo( c2 ) == 0);
    }
    @Test
    public void testCompareToGreater()
    {
        Card c1 = new Card(100);
        Card c2 = new Card(50);
        assertTrue( c1.compareTo( c2 ) > 0);
    }
    @Test
    public void testCompareToLess()
    {
        Card c1 = new Card(100);
        Card c2 = new Card(200);
        assertTrue( c1.compareTo( c2 ) < 0);
    }
    @Test
    public void testGetId()
    {
        Card c = new Card(100);
        assertEquals( c.getId(), 100);
    }
    @Test
    public void testToString()
    {
        Card c = new Card(100);
        assertEquals( c.toString(), "Card [id = " + c.getId() + "]");
```

Networked Zet Game

- Instead of racing against a computer to find the most number of sets possible, you can play with a friend!!!
- Two player game
- Once one player clicks the set button, only they can select a set
  - If it is a set, the set will be removed from both screens, 3 new cards will be added, and the player will be awarded a point
- The player with the most sets wins
- Differences
  - Have to click start on both screens
  - No difficulty since it all depends on your opponent

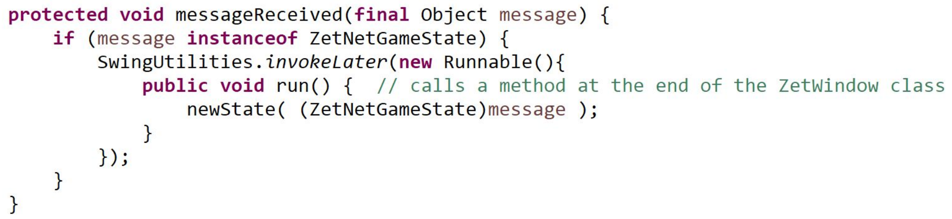
Networking

- The main things that had to be updated between the two clients are:
  - The *ZetNetGameModel* representing the deck of cards and the open cards shown in the GUI
  - An integer array representing the scores of the two players
- Whenever a player makes a move, this is communicated to the other client using *connection.send(ZetNetGameModel)*
- The *ZetNetGameHub* receives it using the *messageReceived()* method and updates the *ZetNetGameState* accordingly



```
protected void messageReceived(int playerID, Object message) {
    state.applyMessage(playerID, message);
    sendToAll(state);
}
```

Hub receiving the message representing the state change on the two clients



```
protected void messageReceived(final Object message) {
    if (message instanceof ZetNetGameState) {
        SwingUtilities.invokeLater(new Runnable(){
            public void run() {  // calls a method at the end of the ZetWindow class
                newState( (ZetNetGameState)message );
            }
        });
    }
}
```

Client receiving the updated state from the hub to synchronize the clients

Classes added for Networking

- Main
- ZetNetGameState
- ZetNetGameHub
- ZetNetGameModel
- ZetNetTableDisplay
- ZetNetTable
- HumanZetNetPlayer

Second Demonstration!!!

Specification Changes

- Originally, we planned on storing the three cards that the user chooses for a set in a LinkedList of ZetCards
    - We decided that this was not practical and it was unnecessary so instead, we stored it in an array
- We added multiple classes in order for the networked game to run
- We had to make all of the classes implement *Serializable* because that was the only way to send the objects of a class via socket for networking

Thank You!!!