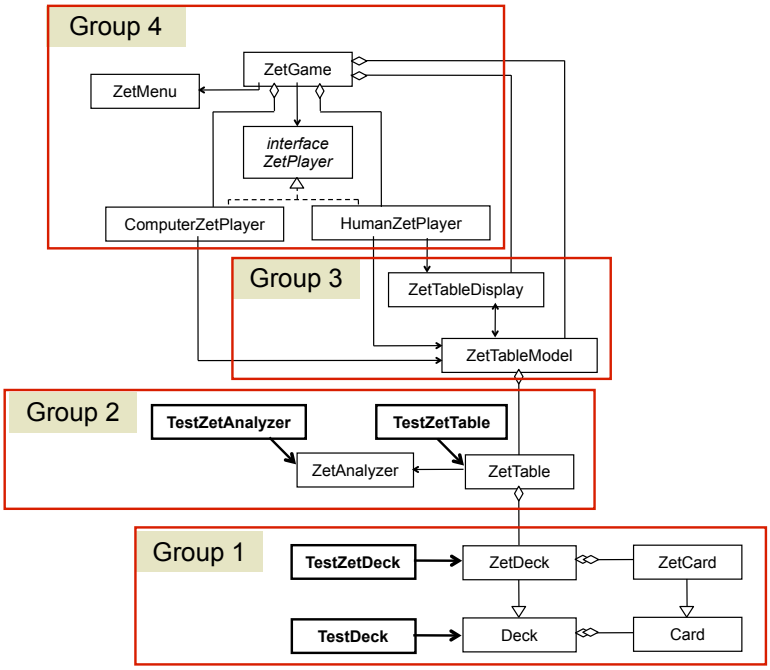
Team Development



Team Development (cont'd)

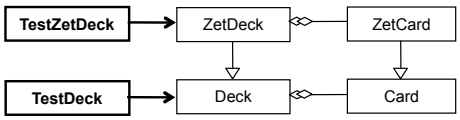
Project Leader

Group 1 (2-6 people)	Group 2 (2-4 people)	Group 3 code supplied (or 2-3 people)	Group 4 code supplied
Card	ZetAnalyzer	ZetTableDisplay	ZetGame
Deck	TestZetAnalyzer	ZetGameModel	ZetMenu
TestDeck	ZetTable		ZetPlayer
ZetCard	TestZetTable		ComputerZetPlayer
ZetDeck			HumanZetPlayer
TestZetDeck			

Developers — Group 1

2-6 people

- Classes:
  - Card
  - Deck
  - TestDeck
  - ZetCard
  - ZetDeck
  - TestZetDeck

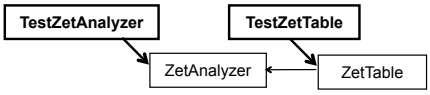


- Required skills:
  - Basic constructors and “get” methods
  - java.util.ArrayList
  - Selection Sort or Collections.sort(...)
  - Inheritance, super(...)

Developers — Group 2

2-4 people

- Classes:
  - ZetTable
  - TestZetTable
  - ZetAnalyzer
  - TestZetAnalyzer

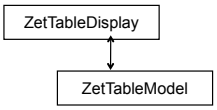


- Required skills:
  - Array algorithms
  - Static methods
  - Modulo arithmetic

Developers — Group 3

Code is Supplied (or 2-3 people)

- Classes:
  - ZetTableDisplay
  - ZetGameModel

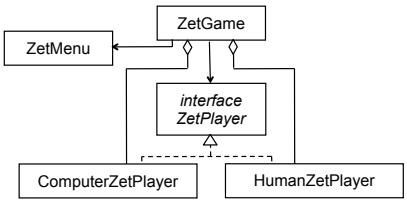


- Required skills:
  - Graphics
  - MVC concept and java.util.Observer / Observable

Developers — Group 4

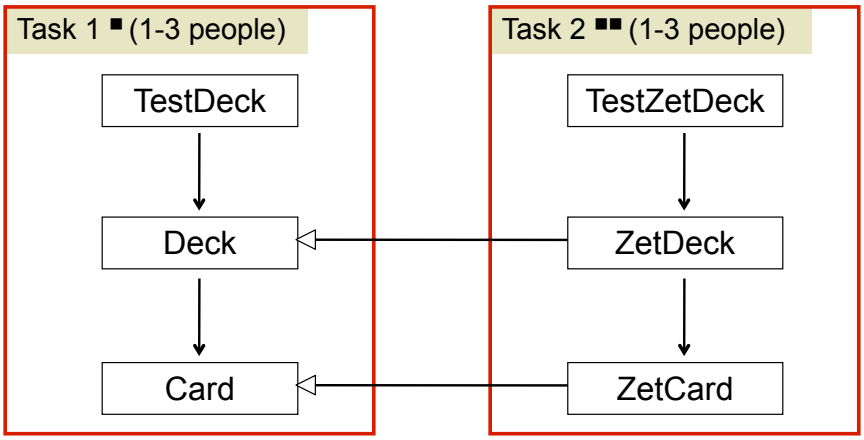
Code is Supplied

- Classes:
  - ZetGame
  - ZetMenu
  - ZetPlayer
  - ComputerZetPlayer
  - HumanZetPlayer



- Prerequisite skills:
  - GUI design
  - javax.swing
  - Mouse, keyboard, and timer event handling

Group 1



Group 1 Task 1-a■

```
public class Card
      implements Comparable<Card>

{
   public Card (int id) { ... }

   public int getId ( ) { ... }
   public boolean equals (Object other)  { ... }
   public int compareTo (Card other)  { ... }
   public String toString ( )  { ... }

   //  Fields:
   private int id;
}
```

Group 1 Task 1-b

public class Deck

```
{
   public Deck ( ) { ... }
   public Deck (int capacity) { ... }      // creates an empty deck
                                           //   of given capacity
   public int getNumCards ( ) { ... }
   public boolean isEmpty ( )  { ... }
   public void add (Card card)  { ... }  // adds card to the top
   public Card takeTop ( )  { ... }      // removes card from the top
   public void shuffle ( );
   public void sort ( );
   public String toString ( )  { ... }

   //  Fields:
   ...
}
```

See implementation tips in **Deck.java**

Group 1 Task 1-c

public class TestDeck

- Create an empty deck
- Add a few cards
- Print out
- Shuffle
- Print out
- Sort
- Print out
- Remove cards one by one;
  print out after each removed card

Group 1 Task 2-a

public class ZetCard
extends Card

```
{
   // Combines the four attributes to make a
   //   unique ID in the range from 0 to 80.
   public ZetCard (int number, int shape,
                                   int fill, int color) { ... }

   public int getNumber ( )  { ... }
   public int getShape ( )  { ... }
   public int getFill ( )  { ... }
   public int getColor ( )  { ... }
   public String toString ( )  { ... }

   // Fields:
   ...
 }
```


Group 1 Task 2-b

public class ZetDeck

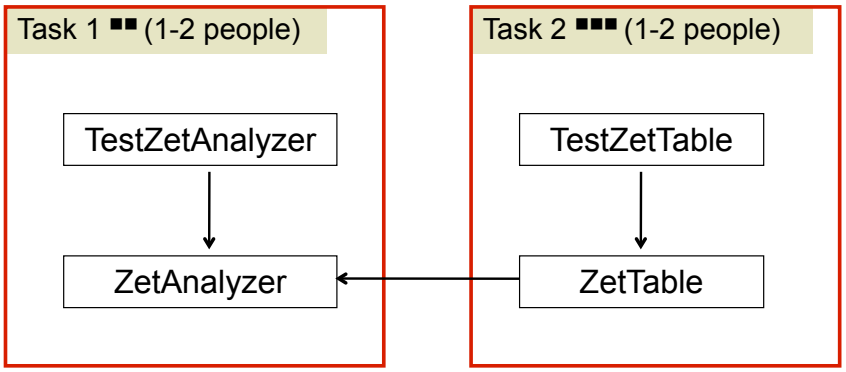
```
{
   public ZetDeck ( ) { ... }      // creates a full deck of
                                   //   81 SET cards
}
```

Group 1 Task 2-c

public class TestZetDeck

- Create a ZetDeck
- Remove and print out three top cards

Group 2



Group 2 Task 1-a■■

public class ZetAnalyzer

```
{
   public static boolean isZet (ZetCard card1,
                        ZetCard card2, ZetCard card3) { ... }

   public static int[ ] findZet (ZetCard[ ] cards) { ... }
}
```

See implementation tips in **ZetAnalyzer.java**

Group 2 Task 1-b

public class TestZetAnalyzer

- Create a ZetDeck
- Open and print out a few cards
- Find and print out all “sets” by calling isZet on all triplets of cards
- Find and print out one “set” by calling findZet

Group 2 Task 2-a

public class ZetTable

```
{
   ...
}
```

See the specs in the javadoc docs and the implementation tips in **ZetTable.java**

public class TestZetTable

- See javadoc documentation for ZetTable.java
- Create a ZetTable object
- Simulate a SET game for one player:
  - Call table.findZet ( ); while a “set” is not found, call table.open3Cards ( ); if it returns false, the game is over
  - Print out the “set”
  - Call table.remove3Cards (...) to remove the “set”
  - If not enough cards open (! table.enoughOpen( )), open 3 more cards; if can’t open, the game is over
  - Repeat the above steps until the deck is empty