Card Class:

```
public class card {

	int suite;
	int value;
	/**
	 * This constructor creates a new card with given values.
	 * @param setsuite The suite that this card will be
	 * @param setvalue The value this card will have
	 */

	public card(int setsuite, int setvalue) {
		value=setvalue;
		suite=setsuite;

	}

	/**
	 * This returns the information on the card in a printable string
	 * I know you said to never use breaks, but hopefully this is an exception
	 * because the switch() method required that you use breaks and I think
	 * this is the most efficient way to go about it
	 * @returns a string of the card data
	 */

	public String toString() {
		String cardval = "Joker";
		String suiteval = "clubs";
		if (value>9) {
			switch(value) {
				case 10: cardval ="jack";
					break;
				case 11: cardval ="queen";
					break;
				case 12: cardval ="king";
					break;
			}
		}
		else {
			cardval=Integer.toString(value+1);
		}

		switch(suite) {
			case 1: suiteval ="hearts";
				break;
			case 2: suiteval ="spades";
				break;
			case 3: suiteval ="diamonds";
				break;
			case 4: suiteval="clubs";
		}

		return cardval + " of " + suiteval;
	}
```

```
	/**
	 * A getter for the suite
	 * @return Returns the suite of the card
	 */
	public int getsuite() {
		return suite;
	}

	/**
	 * A getter for the card value
	 * @return The cards value
	 */
	public int getvalue() {
		return value;
	}

}
```

Deck class:

```
import java.util.Random;

public class deck {
	card[] deck;
	Random Random;
	int dealer;
	int deckitterator;

	/**
	 * The constructor takes no inputs and fills itself with cards.
	 * @param This constructor takes no params
	 */
	deck() {
		deck = new card[52];
		Random = new Random();
		dealer = 0;
		deckitterator=0;
		for(int i=1;i<5;i++) {
			for(int j=0;j<13;j++) {
				deck[deckitterator] = new card(i,j);
				deckitterator++;
			}
		}
	}

	/**
	 * This function evaluates the contents of the deck
	 * @return This returns the contents of the deck in a printable string
	 */
	public String tostring() {
		String alldeck = "The current deck:\n";
		for(int i=0;i<=51;i++){
			alldeck = alldeck + deck[i].toString() + "\n";
```

```
		}
		return alldeck;
	}

	/**
	 *
	 * @return This returns the suite value of a new card
	 */
	public int deal() {
		int returnval=deck[dealer].getsuite();
		dealer++;
		return returnval;

	}
	/**
	 * Gets the dealer value
	 * @return returns the Int dealer value
	 */
	public int getDealerVal() {

		return dealer;
	}

	/**
	 * Shuffles the deck
	 */
	public void shuffle() {
		dealer=0;

		for(int i=0;i<=51;i++){
			int randomval = Random.nextInt(51);
			card firstarr = deck[i];
			card secondarr = deck[randomval];
			deck[i] = secondarr;
			deck[randomval] = firstarr;
		}
	}


}
```

Simulator class:

```
public class simulator {
	int topcard;
	int[] hand;
	deck deck;
	final int DECKSIZE=52;

	/**
	 * Default constructor takes no inputs
	 */
	public simulator() {
		topcard=0;
		hand=new int[DECKSIZE]; // 52 cards in a deck
```

```
		deck=new deck();
	}

	/**
	 * This method goes through and plays the game according to the rules
	 * @return Returns whether the game was a success or not
	 */
	public boolean playgame() {
		boolean outcome = false;
		deck.shuffle(); //Shuffles the cards

		for (int i = 0; i < 4; i++) { // Initiates a game by dealing 4 cards
			hand[i] = deck.deal();
		}

		topcard=3; //Tells the simulator how many cards there are to start

		while (deck.getDealerVal() <= 51) {
			if (topcard >= 3) {
				check();
				topcard++;

				if (topcard >= 3) {
					hand[topcard] = deck.deal();
				}
			}
			else if(topcard<0) {//win condition
				outcome = true;
				return outcome;
			}
			else {
				refill();
			}
		}
		return outcome; //Loss condition

	}

	/**
	 * This method removes the selected range of cards and resorts them so there are
no spaces
	 * @param second Positive range value for what is to be deleted
	 * @param first Positive range value for what is to be deleted
	 */
	private void remove(int second, int first) {
		if (hand[second + 1] != 0) {
			int transfer = hand[second + 1];
			hand[second+1] = 0;
			hand[first] = transfer;
			hand[second] = 0;
			topcard = second-1;
		}

		else {
			for (int i = second; i >= first; i--) {
				hand[i] = 0;
				topcard--;
			}
```

```
                topcard--;
            }
        }

        /**
         * This method checks if any cards can be removed, and if so calls remove()
         */

        private void check() {
            while (topcard>=3 && hand[topcard] == hand[topcard - 3]) {
                if (hand[topcard] == hand[topcard - 2]
                        && hand[topcard] == hand[topcard - 1]) {
                    remove(topcard, (topcard - 3));;
                } else {
                    remove((topcard - 1), (topcard - 2));
                    hand[topcard + 3] = hand[topcard + 1];
                }
            }
        }

        /**
         * This refills the current hand; for use when there are less than four cards
         */
        private void refill() {
            while (topcard < 3 && deck.getDealerVal()<=51) { //Adds cards if there are not
enough
                hand[topcard] = deck.deal();
                topcard++;
            }
        }

}
```

Playgame Class :

```
public class playGame {

        /**
         * This tests out the rules of the solitaire game
         * @author xavier
         *
         *I affirm that I have carried out the attached
         *academic endeavors with full academic honesty, in
         *accordance with the Union College Honor Code and
         *the course syllabus.
         *
         */
        public static void main(String[] args) {
            simulator game = new simulator();
            double score=0.0;
            int tests=1000;
            game.playgame();

            while(tests<=10000) {
                for(int j=0;j<=tests;j++) {
                    if(game.playgame()) {
```

```
                        score++;
                    }
                }

                System.out.println((int)(score) + "/" + tests + " = " + (int)((score/tests)*100) +
"%");
                tests=tests+1000;
                score=0;
            }
        }

}
```