

JAVA TECHNICAL GUIDE

**Created by-
NEELABH SRIVASTAVA**

Definition of Java

Java is a programming language and computing platform first released by Sun Microsystems in 1995. There are lots of applications and websites that will not work unless you have Java installed, and more are created every day. Java is fast, secure, and reliable. From laptops to datacenters, game consoles to scientific supercomputers, cell phones to the Internet, Java is everywhere!

Like English, Java has a set of rules that determine how the instructions are written. These rules are known as its syntax. Once a program has been written, the high-level instructions are translated into numeric codes that computers can understand and execute.

Advantages:

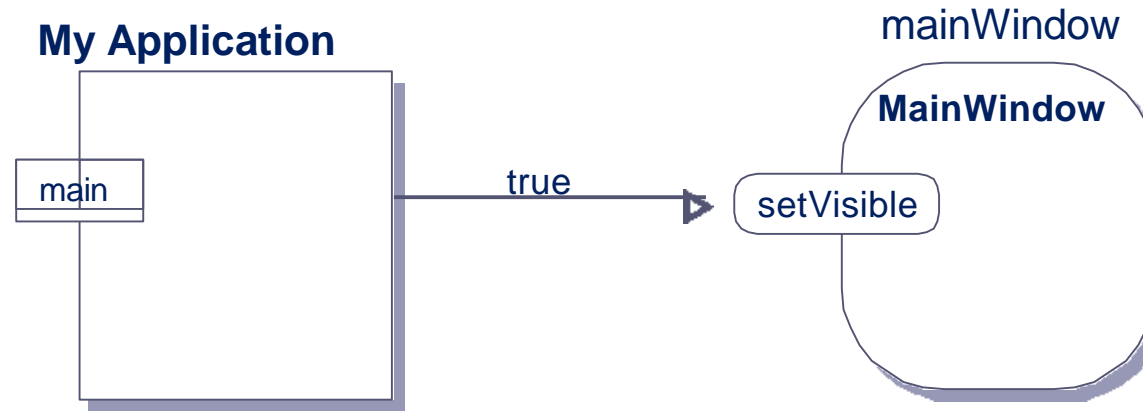
- Ease of Use
- Reliability
- Security
- Platform Independence

Program Components

A Java program is composed of:

- Comments
- Import statements
- class declarations.

Object Diagram



Object Creation

Object Name

Class Name

Argument

mainWindow = new MainWindow ();

Comments Types

```
/*  
    This is a comment with  
    three lines of  
    text.
```

Multiline Comment

```
*/
```

```
// This is a comment  
// This is another comment  
// This is a third comment
```

Single line Comments

```
/**
```

```
 * This class provides basic clock functions. In  
   addition  
 * to reading the current time and today's date, you  
   can  
 * use this class for stopwatch functions.  
 */
```

javadoc Comments

Types Of Variables

Type	Description
byte	8 bit signed integer
short	16 bit signed integer
int	32 bit signed integer
long	64 bit signed integer
float	32 bit signed real number
double	64 bit signed real number
char	16 bit Unicode character (ASCII and beyond)
boolean	1 bit true or false value
String	A sequence of characters between double quotes ("")

Keywords in Java

abstract	boolean	break	byte	case	catch	char
class	const	continue	default	do	double	else
extends	final	finally	float	for	goto	if
implements	import	instanceof	int	interface	long	native
new	package	private	protected	public	return	short
static	super	switch	synchronized	this	throw	throws
transient	try	void	volatile	while		

Project - Introduction

This project is proposed to students in order to evaluate their skills for the fundamental period of Java Programming and UML.

This goal is achieved through the realization of a console application which aims at managing Card Game preparation and execution.

Technical Guide - Java &
UML

MSc - Fall Intake

Specifications

The usual problem while preparing and running an evaluation, is to :

Constitute an appropriate evaluation corresponding of the required level

Reuse former questions

Organize sample evaluations

Correct automatically the MCQ questions.



To handle the most of the possible cases, there are several types of question to consider.

- MCQ Questions
- Open Questions
- Associative Questions

The MCQ questions

The MCQ questions are composed of a question text and a set of possible choices, each choice can be right or wrong. It can also be interesting to add a extra content, like some code extract, some picture or some other kind of media (video, music etc.).

The Open Questions

The open questions are composed only by a question, and some hints, additionally they can be completed by a extra media content.

Associative questions

The associative questions are questions where it necessary to assign some propositions to some descriptions, like in the following.

Common questions attributes

Each question has a some extra attributes to describe the **topic** (tag) and the **difficulty** of the question. Those two fields help to balance the overall exam complexity, and the topics coverage.

Those attributes can be taken in account for automatic exam assembly.

Code

```
import java.util.Random;
import java.util.LinkedList;
import javax.xml.parsers.*;
import javax.xml.transform.*;
import javax.xml.transform.dom.*;
import javax.xml.transform.stream.*;
import org.w3c.dom.*;
import java.io.*;
```

```
public class WarGame {
```

```
    private static int numberOfCards = 52;
    private static int wins[] = new int[]{0,0,0,0};
```

```
    // The four players hands
```

```
    private static LinkedList<String> players[] = new LinkedList[]{
        new LinkedList<String>(), new LinkedList<String>(),
        new LinkedList<String>(), new LinkedList<String>()
    };
```

Code Conti..

```
public static void main(String args[]) {  
  
    try {  
        DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  
        DocumentBuilder db = dbf.newDocumentBuilder();  
        File file = new File("store.xml");  
        file.createNewFile();  
        Document document = db.parse(file);  
        wins[0] = Integer.parseInt(document.getElementsByTagName("Player1").item(0).getFirstChild().getNodeValue());  
        wins[1] = Integer.parseInt(document.getElementsByTagName("Player2").item(0).getFirstChild().getNodeValue());  
        wins[2] = Integer.parseInt(document.getElementsByTagName("Player3").item(0).getFirstChild().getNodeValue());  
        wins[3] = Integer.parseInt(document.getElementsByTagName("Player4").item(0).getFirstChild().getNodeValue());  
    } catch (Exception ex){  
  
    }  
}
```

Code Conti..

```
String colors[] = new String[]{  
    "club", "diamond", "heart", "spade"  
};  
for (String col : colors)  
{  
    for (int i = 2; i <= 10; i++) {  
        System.out.println(i + " of " + col);  
    }  
    for (String card: new String[]{"Jack", "Queen", "King", "Ace"}){  
        System.out.println(card + " of " + col);  
    }  
}  
  
startMatch();
```

Code Conti..

```
}
```

```
public static void persistData(String filename) {  
    Document dom;  
    Element e = null;  
  
    DocumentBuilderFactory dbf = DocumentBuilderFactory.newInstance();  
    try {  
        DocumentBuilder db = dbf.newDocumentBuilder();  
        dom = db.newDocument();  
        Element rootEle = dom.createElement("Players");  
  
        e = dom.createElement("Player1");  
        e.appendChild(dom.createTextNode(wins[0]+""));  
        rootEle.appendChild(e);  
    }  
}
```

Code Conti..

```
Transformer tr = TransformerFactory.newInstance().newTransformer();
    tr.setOutputProperty(OutputKeys.INDENT, "yes");
    tr.setOutputProperty(OutputKeys.METHOD, "xml");
    tr.setOutputProperty(OutputKeys.ENCODING, "UTF-8");
    tr.setOutputProperty("{http://xml.apache.org/xslt}indent-amount", "4");

    tr.transform(new DOMSource(dom),
        new StreamResult(new FileOutputStream(file)));

    } catch (TransformerException te) {
        System.out.println(te.getMessage());
    } catch (IOException ioe) {
        System.out.println(ioe.getMessage());
    }
} catch (ParserConfigurationException pce) {
    System.out.println("Unhandled error");
}
}
```


Code Conti..

```
public static void startMatch(){  
    int counter = 0;
```

```
    LinkedList<String> cards = new LinkedList<String>();  
    for (int i = 1; i <= numberOfCards; i++) {  
        cards.add(""+i);  
    }
```

```
    Random generator = new Random();  
    for (int i = 0; i < numberOfCards; i++) {  
        int index = generator.nextInt(cards.size());  
        String card = cards.get(index);  
        cards.remove(index);  
        players[i%4].add(card);
```

Code Conti..

```
while (!finishedMatch()){
    counter++;
    String roundCards[] = new String[4];
    for (int i = 0; i < 4; i++) {
        if(players[i].size() > 0){
            int index = generator.nextInt(players[i].size());
            String card = players[i].get(index);
            players[i].remove(index);
            roundCards[i] = card;
        }else{
            roundCards[i] = "0";
        }
    }
    System.out.println(roundCards[0] + " " + roundCards[1] + " " + roundCards[2] + " " +
roundCards[3]);
    int roundWinner = getTheBigger(roundCards);
```

Code Conti..

```

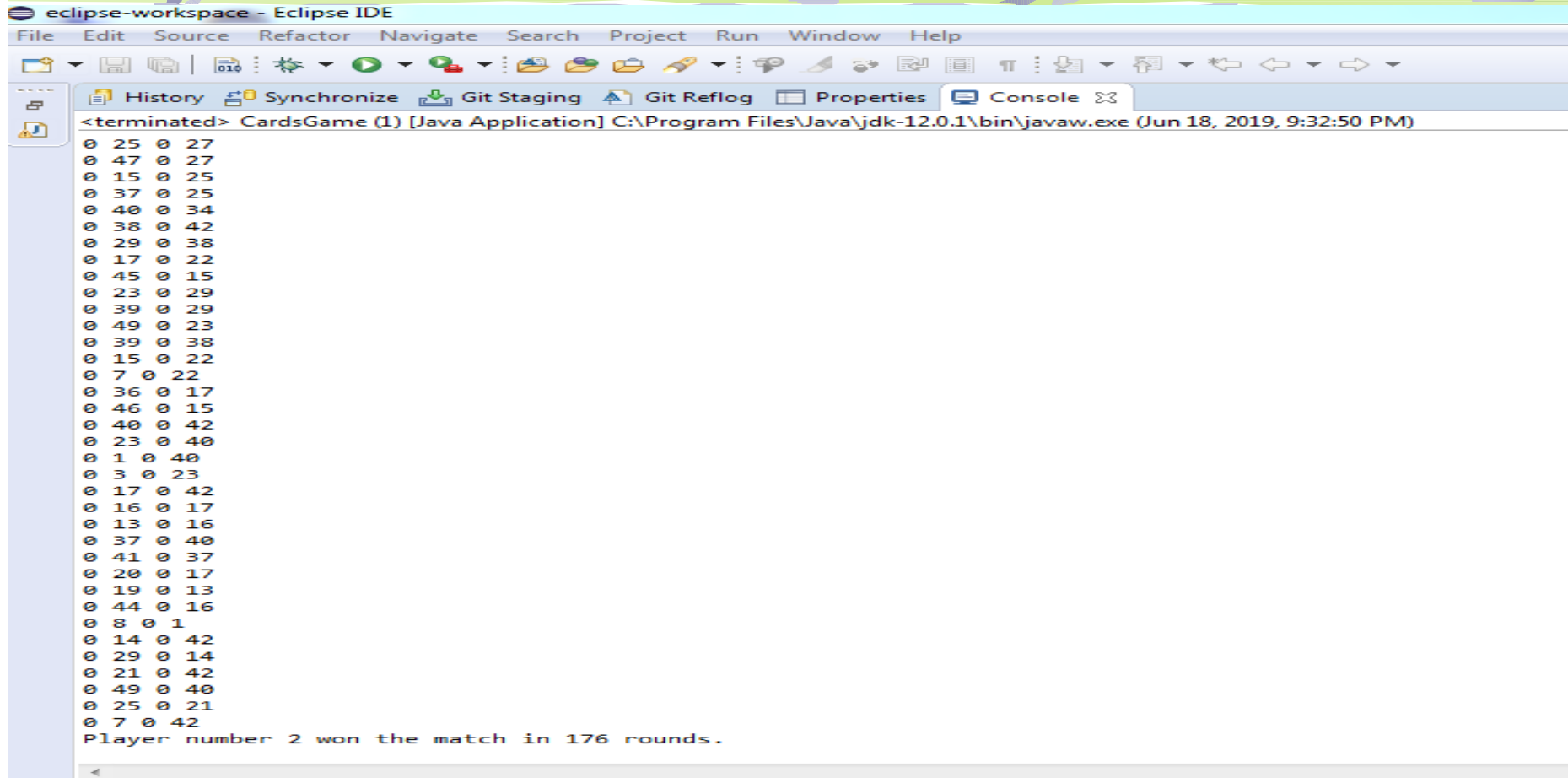
for (String card: roundCards){
    if(card != "0")
        players[roundWinner].add(card);
    }
}

for (int i = 0; i <= 3; i++) {
    LinkedList<String> player = players[i];
    if(player.size() == numberOfCards){
        System.out.println("Player number " + (i+1) +" won the match in " + counter + "
rounds.");
        wins[i]+=1;
    }
}

persistData("store.xml");
}
}

```

Output



The screenshot shows the Eclipse IDE interface with the console window open. The console title is "<terminated> CardsGame (1) [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (Jun 18, 2019, 9:32:50 PM)". The output consists of 40 lines of card game results, each showing two cards separated by a space. The final line states: "Player number 2 won the match in 176 rounds."

```
<terminated> CardsGame (1) [Java Application] C:\Program Files\Java\jdk-12.0.1\bin\javaw.exe (Jun 18, 2019, 9:32:50 PM)
0 25 0 27
0 47 0 27
0 15 0 25
0 37 0 25
0 40 0 34
0 38 0 42
0 29 0 38
0 17 0 22
0 45 0 15
0 23 0 29
0 39 0 29
0 49 0 23
0 39 0 38
0 15 0 22
0 7 0 22
0 36 0 17
0 46 0 15
0 40 0 42
0 23 0 40
0 1 0 40
0 3 0 23
0 17 0 42
0 16 0 17
0 13 0 16
0 37 0 40
0 41 0 37
0 20 0 17
0 19 0 13
0 44 0 16
0 8 0 1
0 14 0 42
0 29 0 14
0 21 0 42
0 49 0 40
0 25 0 21
0 7 0 42
Player number 2 won the match in 176 rounds.
```

Explanation

In line 21 of the code we find the main () method, and between line 24 and 37 we check if the file store.xml exists, using DocumentBuilderFactory to read from this file the persisted victories of previous games with the method persistData() that we can find in line 72, method that also uses DocumentBuilderFactory to update the file store.xml after each game (for this use the wins[] array that is updated appropriately in the code). Between lines 38 and 50 the letters and their colors are printed in the order ordered in the orientation nesting cycles. Then a call is made to the startMatch() method, whose definition we find in line 123. This method initializes the hands of the players as linked lists (this data structure was selected for its efficiency when inserting and deleting elements), and between lines 131 and 137 all the cards are distributed among these 4 players, then between lines 139 and 158 we have a while statement, that while the game is not finished (the method finishedMatch() checks that any player has all the cards in your hand), select the 4 cards of the round (line 141 to 151), using the linked lists. Then the getTheBigger() method returns the winner of the round and between lines 154 and 157 all the cards in the round are added to the winner's hand. Then between lines 160 and 166, the person who wins is checked and printed, to finally persist the information with the method persistData() that we already analyzed at the beginning.

THANK
YOU