1. Prefer returning Empty Collections instead of Null

If a program is returning a collection which does not have any value, make sure an Empty collection is returned rather than Null elements. This saves a lot of “*if else*” testing on Null Elements.

|  |  |
| --- | --- |
| 1  2  3 | public class getLocationName {      return (null==cityName ? "": cityName);  } |

2. Use Strings carefully

If two Strings are concatenated using “+” operator in a “for” loop, then it creates a new String Object, every time. This causes wastage of memory and increases performance time. Also, while instantiating a String Object, constructors should be avoided and instantiation should happen directly. For example:

|  |  |
| --- | --- |
| 1  2  3  4  5 | //Slower Instantiation  String bad = new String("Yet another string object");    //Faster Instantiation  String good = "Yet another string object" |

3. Avoid unnecessary Objects

One of the most expensive operations (in terms of Memory Utilization) in Java is Object Creation. Thus it is recommended that Objects should only be created or initialized if necessary. Following code gives an example:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | import java.util.ArrayList;  import java.util.List;    public class Employees {        private List Employees;        public List getEmployees() {            //initialize only when required          if(null == Employees) {              Employees = new ArrayList();          }          return Employees;      }  } |

4. Dilemma between Array and ArrayList

Developers often find it difficult to decide if they should go for Array type data structure of ArrayList type. They both have their strengths and weaknesses. The choice really depends on the requirements.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import java.util.ArrayList;    public class arrayVsArrayList {        public static void main(String[] args) {          int[] myArray = new int[6];          myArray[7]= 10; // ArraysOutOfBoundException            //Declaration of ArrayList. Add and Remove of elements is easy.          ArrayList<Integer> myArrayList = new ArrayList<>();          myArrayList.add(1);          myArrayList.add(2);          myArrayList.add(3);          myArrayList.add(4);          myArrayList.add(5);          myArrayList.remove(0);            for(int i = 0; i < myArrayList.size(); i++) {          System.out.println("Element: " + myArrayList.get(i));          }            //Multi-dimensional Array          int[][][] multiArray = new int [3][3][3];      }  } |

1. Arrays have fixed size but ArrayLists have variable sizes. Since the size of Array is fixed, the memory gets allocated at the time of declaration of Array type variable. Hence, Arrays are very fast. On the other hand, if we are not aware of the size of the data, then ArrayList is More data will lead to ArrayOutOfBoundException and less data will cause wastage of storage space.
2. It is much easier to Add or Remove elements from ArrayList than Array
3. Array can be multi-dimensional but ArrayList can be only one dimension.

5. When Finally does not get executed with Try

Consider following code snippet:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | public class shutDownHooksDemo {      public static void main(String[] args) {          for(int i=0;i<5;i++)          {              try {                  if(i==4) {                      System.out.println("Inside Try Block.Exiting without executing Finally block.");                      System.exit(0);                  }              }              finally {                  System.out.println("Inside Finally Block.");              }          }      }  } |

From the program, it looks like “println” inside *finally*block will be executed 5 times. But if the program is executed, the user will find that *finally* block is called only 4 times. In the fifth iteration, *exit* function is called and *finally* never gets called the fifth time. The reason is- System.exit halts execution of all the running threads including the current one. Even *finally* block does not get executed after *try* when *exit* is executed.

When *System.exit* is called, JVM performs two cleanup tasks before shut down:

First, it executes all the *shutdown hooks*which have been registered with *Runtime.addShutdownHook*. This is very useful because it releases the resources external to JVM.

Second is related to *Finalizers*. Either *System.runFinalizersOnExit*or *Runtime.runFinalizersOnExit*. The use of *finalizers*has been deprecated from a long time. *Finalizers*can run on live objects while they are being manipulated by other threads.This results in undesirable results or even in a deadlock.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | public class shutDownHooksDemo {        public static void main(String[] args) {              for(int i=0;i<5;i++)              {                      final int final\_i = i;                      try {                              Runtime.getRuntime().addShutdownHook(                                              new Thread() {                                              public void run() {                                              if(final\_i==4) {                                              System.out.println("Inside Try Block.Exiting without executing Finally block.");                                              System.exit(0);                                              }                                              }                                              });                      }                      finally {                              System.out.println("Inside Finally Block.");                      }                }      }  } |

6. Check Oddity

Have a look at the lines of code below and determine if they can be used to precisely identify if a given number is Odd?

|  |  |
| --- | --- |
| 1  2  3 | public boolean oddOrNot(int num) {      return num % 2 == 1;  } |

These lines seem correct but they will return incorrect results one of every four times (Statistically speaking). Consider a negative Odd number, the remainder of division with 2 will not be 1. So, the returned result will be false which is incorrect!

This can be fixed as follows:

|  |  |
| --- | --- |
| 1  2  3 | public boolean oddOrNot(int num) {      return (num & 1) != 0;  } |

Using this code, not only is the problem of negative odd numbers solved, but this code is also highly optimized. Since, Arithmetic and Logical operations are much faster compared to division and multiplication, the results are achieved faster so in second snippet.

7. Difference between single quotes and double quotes

|  |  |
| --- | --- |
| 1  2  3  4  5  6 | public class Haha {      public static void main(String args[]) {      System.out.print("H" + "a");      System.out.print('H' + 'a');      }  } |

From the code, it would seem return “HaHa” is returned, but it actually returns Ha169. The reason is that if double quotes are used, the characters are treated as a string but in case of single quotes, the char -valued operands ( ‘H’ and ‘a’ ) to int values through a process known as widening primitive conversion. After integer conversion, the numbers are added and return 169.

8. Avoiding Memory leaks by simple tricks

Memory leaks often cause performance degradation of software. Since, Java manages memory automatically, the developers do not have much control. But there are still some standard practices which can be used to protect from memory leakages.

* Always release database connections when querying is complete.
* Try to use Finally block as often possible.
* Release instances stored in Static Tables.

9. Avoiding Deadlocks in Java

Deadlocks can occur for many different reasons. There is no single recipe to avoid deadlocks. Normally deadlocks occur when one synchronized object is waiting for lock on resources locked by another synchronized object.

Try running the below program. This program demonstrates a Deadlock. This deadlock arises because both the threads are waiting for the resources which are grabbed by other thread. They both keep waiting and no one releases.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44 | public class DeadlockDemo {     public static Object addLock = new Object();     public static Object subLock = new Object();       public static void main(String args[]) {          MyAdditionThread add = new MyAdditionThread();        MySubtractionThread sub = new MySubtractionThread();        add.start();        sub.start();     }  private static class MyAdditionThread extends Thread {        public void run() {           synchronized (addLock) {          int a = 10, b = 3;          int c = a + b;              System.out.println("Addition Thread: " + c);              System.out.println("Holding First Lock...");              try { Thread.sleep(10); }              catch (InterruptedException e) {}              System.out.println("Addition Thread: Waiting for AddLock...");              synchronized (subLock) {                 System.out.println("Threads: Holding Add and Sub Locks...");              }           }        }     }     private static class MySubtractionThread extends Thread {        public void run() {           synchronized (subLock) {          int a = 10, b = 3;          int c = a - b;              System.out.println("Subtraction Thread: " + c);              System.out.println("Holding Second Lock...");              try { Thread.sleep(10); }              catch (InterruptedException e) {}              System.out.println("Subtraction  Thread: Waiting for SubLock...");              synchronized (addLock) {                 System.out.println("Threads: Holding Add and Sub Locks...");              }           }        }     }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7 | =====  Addition Thread: 13  Subtraction Thread: 7  Holding First Lock...  Holding Second Lock...  Addition Thread: Waiting for AddLock...  Subtraction  Thread: Waiting for SubLock... |

But if the order in which the threads are called is changed, the deadlock problem is resolved.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47 | public class DeadlockSolutionDemo {     public static Object addLock = new Object();     public static Object subLock = new Object();       public static void main(String args[]) {          MyAdditionThread add = new MyAdditionThread();        MySubtractionThread sub = new MySubtractionThread();        add.start();        sub.start();     }      private static class MyAdditionThread extends Thread {        public void run() {           synchronized (addLock) {          int a = 10, b = 3;          int c = a + b;              System.out.println("Addition Thread: " + c);              System.out.println("Holding First Lock...");              try { Thread.sleep(10); }              catch (InterruptedException e) {}              System.out.println("Addition Thread: Waiting for AddLock...");              synchronized (subLock) {                 System.out.println("Threads: Holding Add and Sub Locks...");              }           }        }     }       private static class MySubtractionThread extends Thread {        public void run() {           synchronized (addLock) {          int a = 10, b = 3;          int c = a - b;              System.out.println("Subtraction Thread: " + c);              System.out.println("Holding Second Lock...");              try { Thread.sleep(10); }              catch (InterruptedException e) {}              System.out.println("Subtraction  Thread: Waiting for SubLock...");              synchronized (subLock) {                 System.out.println("Threads: Holding Add and Sub Locks...");              }           }        }     }  } |

Output:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9 | =====  Addition Thread: 13  Holding First Lock...  Addition Thread: Waiting for AddLock...  Threads: Holding Add and Sub Locks...  Subtraction Thread: 7  Holding Second Lock...  Subtraction  Thread: Waiting for SubLock...  Threads: Holding Add and Sub Locks... |

10. Reserve memory for Java

Some of the Java applications can be highly CPU intensive as well as they need a lot of RAM. Such applications generally run slow because of a high RAM requirement. In order to improve performance of such applications, RAM is reserved for Java. So, for example, if we have a Tomcat webserver and it has 10 GB of RAM. If we like, we can allocate RAM for Java on this machine using the following command:

|  |  |
| --- | --- |
| 1 | export JAVA\_OPTS="$JAVA\_OPTS -Xms5000m -Xmx6000m -XX:PermSize=1024m -XX:MaxPermSize=2048m" |

* Xms = Minimum memory allocation pool
* Xmx = Maximum memory allocation pool
* XX:PermSize = Initial size that will be allocated during startup of the JVM
* XX:MaxPermSize = Maximum size that can be allocated during startup of the JVM

11. How to time operations in Java

There are two standard ways to time operations in Java: **System.currentTimeMillis()**and **System.nanoTime()** The question is, which of these to choose and under what circumstances. In principle, they both perform the same action but are different in the following ways:

1. System.currentTimeMillis takes somewhere between 1/1000th of a second to 15/1000th of a second (depending on the system) but System.nanoTime() takes around 1/1000,000th of a second (1,000 nanos)
2. System.currentTimeMillis takes a few clock cycles to perform Read Operation. On the other hand, System.nanoTime() takes 100+ clock cycles.
3. System.currentTimeMillis reflects Absolute Time (Number of millis since 1 Jan 1970 00:00 (Epoch Time)) but System.nanoTime() does not necessarily represent any reference point.

12. Choice between Float and Double

|  |  |  |
| --- | --- | --- |
| **Data type** | **Bytes used** | **Significant figures (decimal)** |
| Float | 4 | 7 |
| Double | 8 | 15 |

Double is often preferred over float in software where precision is important because of the following reasons:

Most processors take nearly the same amount of processing time to perform operations on Float and Double. Double offers far more precision in the same amount of computation time.

13. Computation of power

To compute power (^), java performs Exclusive OR (XOR). In order to compute power, Java offers two options:

1. **Multiplication:**

|  |  |
| --- | --- |
| 1  2  3  4  5 | double square = double a \* double a;                            // Optimized  double cube = double a \* double a \* double a;                   // Non-optimized  double cube = double a \* double square;                         // Optimized  double quad = double a \* double a \* double a \* double a;            // Non-optimized  double quad = double square \* double square;                    // Optimized |

1. **pow(double base, double exponent):**‘pow’ method is used to calculate where multiplication is not possible (base^exponent)

|  |  |
| --- | --- |
| 1 | double cube = Math.pow(base, exponent); |

Math.pow should be used ONLY when necessary. For example, exponent is a fractional value. That is because Math.pow() method is typically around 300-600 times slower than a multiplication.

14. How to handle Null Pointer Exceptions

Null Pointer Exceptions are quite common in Java. This exception occurs when we try to call a method on a Null Object Reference. For example,

|  |  |
| --- | --- |
| 1 | int noOfStudents = school.listStudents().count; |

If in the above example, if get a NullPointerException, then either school is null or listStudents() is Null. It’s a good idea to check Nulls early so that they can be eliminated.

|  |  |
| --- | --- |
| 1  2  3  4 | private int getListOfStudents(File[] files) {        if (files == null)          throw new NullPointerException("File list cannot be null");      } |

15. Encode in JSON

JSON (JavaScript Object Notation) is syntax for storing and exchanging data. JSON is an easier-to-use alternative to XML. Json is becoming very popular over internet these days because of its properties and light weight. A normal data structure can be encoded into JSON and shared across web pages easily. Before beginning to write code, a JSON parser has to be installed. In below examples, we have used json.simple (https://code.google.com/p/json-simple/).

Below is a basic example of Encoding into JSON:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21 | import org.json.simple.JSONObject;  import org.json.simple.JSONArray;    public class JsonEncodeDemo {        public static void main(String[] args) {            JSONObject obj = new JSONObject();          obj.put("Novel Name", "Godaan");          obj.put("Author", "Munshi Premchand");            JSONArray novelDetails = new JSONArray();          novelDetails.add("Language: Hindi");          novelDetails.add("Year of Publication: 1936");          novelDetails.add("Publisher: Lokmanya Press");            obj.put("Novel Details", novelDetails);            System.out.print(obj);      }  } |

Output:

|  |  |
| --- | --- |
| 1 | {"Novel Name":"Godaan","Novel Details":["Language: Hindi","Year of Publication: 1936","Publisher: Lokmanya Press"],"Author":"Munshi Premchand"} |

16. Decode from JSON

In order to decode JSON, the developer must be aware of the schema. The details can be found in below example:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74 | import java.io.FileNotFoundException;  import java.io.FileReader;  import java.io.IOException;  import java.util.Iterator;    import org.json.simple.JSONArray;  import org.json.simple.JSONObject;  import org.json.simple.parser.JSONParser;  import org.json.simple.parser.ParseException;    public class JsonParseTest {        private static final String filePath = "//home//user//Documents//jsonDemoFile.json";        public static void main(String[] args) {            try {              // read the json file              FileReader reader = new FileReader(filePath);              JSONParser jsonParser = new JSONParser();              JSONObject jsonObject = (JSONObject)jsonParser.parse(reader);                // get a number from the JSON object              Long id =  (Long) jsonObject.get("id");              System.out.println("The id is: " + id);                // get a String from the JSON object              String  type = (String) jsonObject.get("type");              System.out.println("The type is: " + type);                // get a String from the JSON object              String  name = (String) jsonObject.get("name");              System.out.println("The name is: " + name);                // get a number from the JSON object              Double ppu =  (Double) jsonObject.get("ppu");              System.out.println("The PPU is: " + ppu);                // get an array from the JSON object              System.out.println("Batters:");              JSONArray batterArray= (JSONArray) jsonObject.get("batters");              Iterator i = batterArray.iterator();              // take each value from the json array separately              while (i.hasNext()) {                  JSONObject innerObj = (JSONObject) i.next();                  System.out.println("ID "+ innerObj.get("id") +                          " type " + innerObj.get("type"));              }                // get an array from the JSON object              System.out.println("Topping:");              JSONArray toppingArray= (JSONArray) jsonObject.get("topping");              Iterator j = toppingArray.iterator();              // take each value from the json array separately              while (j.hasNext()) {                  JSONObject innerObj = (JSONObject) j.next();                  System.out.println("ID "+ innerObj.get("id") +                          " type " + innerObj.get("type"));              }              } catch (FileNotFoundException ex) {              ex.printStackTrace();          } catch (IOException ex) {              ex.printStackTrace();          } catch (ParseException ex) {              ex.printStackTrace();          } catch (NullPointerException ex) {              ex.printStackTrace();          }        }    } |

**jsonDemoFile.json**

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23 | {      "id": 0001,      "type": "donut",      "name": "Cake",      "ppu": 0.55,      "batters":          [              { "id": 1001, "type": "Regular" },              { "id": 1002, "type": "Chocolate" },              { "id": 1003, "type": "Blueberry" },              { "id": 1004, "type": "Devil's Food" }          ],      "topping":          [              { "id": 5001, "type": "None" },              { "id": 5002, "type": "Glazed" },              { "id": 5005, "type": "Sugar" },              { "id": 5007, "type": "Powdered Sugar" },              { "id": 5006, "type": "Chocolate with Sprinkles" },              { "id": 5003, "type": "Chocolate" },              { "id": 5004, "type": "Maple" }          ]  } |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17 | The id is: 1  The type is: donut  The name is: Cake  The PPU is: 0.55  Batters:  ID 1001 type Regular  ID 1002 type Chocolate  ID 1003 type Blueberry  ID 1004 type Devil's Food  Topping:  ID 5001 type None  ID 5002 type Glazed  ID 5005 type Sugar  ID 5007 type Powdered Sugar  ID 5006 type Chocolate with Sprinkles  ID 5003 type Chocolate  ID 5004 type Maple |

17. Simple String Search

Java offers a Library method called indexOf(). This method is used with String Object and it returns the position of index of desired string. If the string is not found then -1 is returned.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13 | public class StringSearch {        public static void main(String[] args) {          String myString = "I am a String!";            if(myString.indexOf("String") == -1) {              System.out.println("String not Found!");          }          else {              System.out.println("String found at: " + myString.indexOf("String"));          }      }  } |

18. Listing content of a directory

In order to list the contents of a directory, below program can be used. This program simply receives the names of the all sub-directory and files in a folder in an Array and then that array is sequentially traversed to list all the contents.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14 | import java.io.\*;    public class ListContents {      public static void main(String[] args) {          File file = new File("//home//user//Documents/");          String[] files = file.list();            System.out.println("Listing contents of " + file.getPath());          for(int i=0 ; i < files.length ; i++)          {              System.out.println(files[i]);          }      }  } |

19. A Simple IO

In order to read from a file and write to a file, Java offers FileInputStream and FileOutputStream Classes. FileInputStream’s constructor accepts filepath of Input File as argument and creates File Input Stream. Similarly, FileOutputStream’s constructor accepts filepath of Output File as argument and creates File Output Stream.After the file handling is done, it’s important to “close” the streams.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25 | import java.io.\*;    public class myIODemo {      public static void main(String args[]) throws IOException {          FileInputStream in = null;          FileOutputStream out = null;            try {              in = new FileInputStream("//home//user//Documents//InputFile.txt");              out = new FileOutputStream("//home//user//Documents//OutputFile.txt");                int c;              while((c = in.read()) != -1) {                  out.write(c);              }          } finally {              if(in != null) {                  in.close();              }              if(out != null) {                  out.close();              }          }      }  } |

20. Executing a shell command from Java

Java offers Runtime class to execute Shell Commands. Since these are external commands, exception handling is really important. In below example, we illustrate this with a simple example. We are trying to open a PDF file from Shell command.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29 | import java.io.BufferedReader;  import java.io.InputStream;  import java.io.InputStreamReader;    public class ShellCommandExec {        public static void main(String[] args) {          String gnomeOpenCommand = "gnome-open //home//user//Documents//MyDoc.pdf";            try {              Runtime rt = Runtime.getRuntime();              Process processObj = rt.exec(gnomeOpenCommand);                InputStream stdin = processObj.getErrorStream();              InputStreamReader isr = new InputStreamReader(stdin);              BufferedReader br = new BufferedReader(isr);                String myoutput = "";                while ((myoutput=br.readLine()) != null) {                  myoutput = myoutput+"\n";              }              System.out.println(myoutput);          }          catch (Exception e) {              e.printStackTrace();          }      }  } |

21. Using Regex

Summary of Regular Expression Constructs (Source: Oracle Website)

|  |  |
| --- | --- |
| **Characters** | |
| x | The character x |
| \\ | The backslash character |
| \0n | The character with octal value 0n (0 <= n <= 7) |
| \0nn | The character with octal value 0nn (0 <= n <= 7) |
| \0mnn | The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7) |
| \xhh | The character with hexadecimal value 0xhh |
| \uhhhh | The character with hexadecimal value 0xhhhh |
| \x{h…h} | The character with hexadecimal value 0xh…h (Character.MIN\_CODE\_POINT <= 0xh…h <= Character.MAX\_CODE\_POINT) |
| \t | The tab character (‘\u0009’) |
| \n | The newline (line feed) character (‘\u000A’) |
| \r | The carriage-return character (‘\u000D’) |
| \f | The form-feed character (‘\u000C’) |
| \a | The alert (bell) character (‘\u0007’) |
| \e | The escape character (‘\u001B’) |
| \cx | The control character corresponding to x |

|  |  |
| --- | --- |
| **Character classes** | |
| [abc] | a, b, or c (simple class) |
| [^abc] | Any character except a, b, or c (negation) |
| [a-zA-Z] | a through z or A through Z, inclusive (range) |
| [a-d[m-p]] | a through d, or m through p: [a-dm-p] (union) |
| [a-z&&[def]] | d, e, or f (intersection) |
| [a-z&&[^bc]] | a through z, except for b and c: [ad-z] (subtraction) |
| [a-z&&[^m-p]] | a through z, and not m through p: [a-lq-z](subtraction) |

|  |  |
| --- | --- |
| **Predefined character classes** | |
| . | [Any character (may or may not match line terminators)](https://www.javacodegeeks.com/2015/06/java-programming-tips-best-practices-beginners.html#/home/shubhmoy/Documents\x/lt) |
| \d | A digit: [0-9] |
| \D | A non-digit: [^0-9] |
| \s | A whitespace character: [ \t\n\x0B\f\r] |
| \S | A non-whitespace character: [^\s] |
| \w | A word character: [a-zA-Z\_0-9] |
| \W | A non-word character: [^\w] |

|  |  |
| --- | --- |
| **Boundary matchers** | |
| ^ | The beginning of a line |
| $ | The end of a line |
| \b | A word boundary |
| \B | A non-word boundary |
| \A | The beginning of the input |
| \G | The end of the previous match |
| \Z | [The end of the input but for the final terminator, if any](https://www.javacodegeeks.com/2015/06/java-programming-tips-best-practices-beginners.html#/home/shubhmoy/Documents\x/lt) |
| \z | The end of the input |

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30 | import java.util.regex.Matcher;  import java.util.regex.Pattern;    public class RegexMatches  {      private static String pattern =  "^[\_A-Za-z0-9-]+(\\.[\_A-Za-z0-9-]+)\*@[A-Za-z0-9]+(\\.[A-Za-z0-9]+)\*(\\.[A-Za-z]{2,})$";      private static Pattern mypattern = Pattern.compile(pattern);        public static void main( String args[] ){            String valEmail1 = "testemail@domain.com";          String invalEmail1 = "....@domain.com";          String invalEmail2 = ".$$%%@domain.com";          String valEmail2 = "test.email@domain.com";            System.out.println("Is Email ID1 valid? "+validateEMailID(valEmail1));          System.out.println("Is Email ID1 valid? "+validateEMailID(invalEmail1));          System.out.println("Is Email ID1 valid? "+validateEMailID(invalEmail2));          System.out.println("Is Email ID1 valid? "+validateEMailID(valEmail2));        }        public static boolean validateEMailID(String emailID) {          Matcher mtch = mypattern.matcher(emailID);          if(mtch.matches()){              return true;          }          return false;      }  } |

22. Simple Java Swing Example

With the help of Java Swing GUI can be created. Java offers Javax which contains “swing”. The GUI using swing begin with extending JFrame. Boxes are added so they can contain GUI components like Button, Radio Button, Text box, etc. These boxes are set on top of Container.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33 | import java.awt.\*;  import javax.swing.\*;    public class SwingsDemo extends JFrame  {      public SwingsDemo()      {          String path = "//home//user//Documents//images";          Container contentPane = getContentPane();          contentPane.setLayout(new FlowLayout());            Box myHorizontalBox = Box. createHorizontalBox();          Box myVerticleBox = Box. createVerticalBox();            myHorizontalBox.add(new JButton("My Button 1"));          myHorizontalBox.add(new JButton("My Button 2"));          myHorizontalBox.add(new JButton("My Button 3"));            myVerticleBox.add(new JButton(new ImageIcon(path + "//Image1.jpg")));          myVerticleBox.add(new JButton(new ImageIcon(path + "//Image2.jpg")));          myVerticleBox.add(new JButton(new ImageIcon(path + "//Image3.jpg")));            contentPane.add(myHorizontalBox);          contentPane.add(myVerticleBox);            pack();          setVisible(true);      }        public static void main(String args[]) {          new SwingsDemo();      }  } |

23. Play a sound with Java

Playing sound is a common requirement in Java, especially along with Games.

This demo explains how to play an Audio file along with Java code.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35 | import java.io.\*;  import java.net.URL;  import javax.sound.sampled.\*;  import javax.swing.\*;    // To play sound using Clip, the process need to be alive.  // Hence, we use a Swing application.  public class playSoundDemo extends JFrame {       // Constructor     public playSoundDemo() {        this.setDefaultCloseOperation(JFrame.EXIT\_ON\_CLOSE);        this.setTitle("Play Sound Demo");        this.setSize(300, 200);        this.setVisible(true);          try {           URL url = this.getClass().getResource("MyAudio.wav");           AudioInputStream audioIn = AudioSystem.getAudioInputStream(url);           Clip clip = AudioSystem.getClip();           clip.open(audioIn);           clip.start();        } catch (UnsupportedAudioFileException e) {           e.printStackTrace();        } catch (IOException e) {           e.printStackTrace();        } catch (LineUnavailableException e) {           e.printStackTrace();        }     }       public static void main(String[] args) {        new playSoundDemo();     }  } |

24. PDF Export

Export a table to PDF is a common requirement in Java programs. Using itextpdf, it becomes really easy to export PDF.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36 | import java.io.FileOutputStream;  import com.itextpdf.text.Document;  import com.itextpdf.text.Paragraph;  import com.itextpdf.text.pdf.PdfPCell;  import com.itextpdf.text.pdf.PdfPTable;  import com.itextpdf.text.pdf.PdfWriter;    public class DrawPdf {          public static void main(String[] args) throws Exception {          Document document = new Document();          PdfWriter.getInstance(document, new FileOutputStream("Employee.pdf"));          document.open();            Paragraph para = new Paragraph("Employee Table");          para.setSpacingAfter(20);          document.add(para);            PdfPTable table = new PdfPTable(3);          PdfPCell cell = new PdfPCell(new Paragraph("First Name"));            table.addCell(cell);          table.addCell("Last Name");          table.addCell("Gender");          table.addCell("Ram");          table.addCell("Kumar");          table.addCell("Male");          table.addCell("Lakshmi");          table.addCell("Devi");          table.addCell("Female");            document.add(table);            document.close();        }      } |

25. Sending Email from Java Code

Sending email from Java is simple. We need to install Java Mail Jar and set its path in our program’s classpath. The basic properties are set in the code and we are good to send email as mentioned in the code below:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32 | import java.util.\*;  import javax.mail.\*;  import javax.mail.internet.\*;    public class SendEmail  {      public static void main(String [] args)      {          String to = "recipient@gmail.com";          String from = "sender@gmail.com";          String host = "localhost";            Properties properties = System.getProperties();          properties.setProperty("mail.smtp.host", host);          Session session = Session.getDefaultInstance(properties);            try{              MimeMessage message = new MimeMessage(session);              message.setFrom(new InternetAddress(from));                message.addRecipient(Message.RecipientType.TO,new InternetAddress(to));                message.setSubject("My Email Subject");              message.setText("My Message Body");              Transport.send(message);              System.out.println("Sent successfully!");          }          catch (MessagingException ex) {              ex.printStackTrace();          }      }  } |

26. Measuring time

Many applications require a very precise time measurement. For this purpose, Java provides static methods in System class:

1. **currentTimeMillis():** Returns current time in MilliSeconds since Epoch Time, in Long.

|  |  |
| --- | --- |
| 1  2 | long startTime = System.currentTimeMillis();  long estimatedTime = System.currentTimeMillis() - startTime; |

1. **nanoTime():** Returns the current value of the most precise available system timer, in nanoseconds, in long. nanoTime() is meant for measuring relative time interval instead of providing absolute timing.

|  |  |
| --- | --- |
| 1  2 | long startTime = System.nanoTime();  long estimatedTime = System.nanoTime() - startTime; |

27. Rescale Image

An image can rescaled usingAffineTransform. First of all, Image Buffer of input image is created and then scaled image is rendered.

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16 | import java.awt.Graphics2D;  import java.awt.geom.AffineTransform;  import java.awt.image.BufferedImage;  import java.io.File;  import javax.imageio.ImageIO;    public class RescaleImage {    public static void main(String[] args) throws Exception {      BufferedImage imgSource = ImageIO.read(new File("images//Image3.jpg"));      BufferedImage imgDestination = new BufferedImage(100, 100, BufferedImage.TYPE\_INT\_RGB);      Graphics2D g = imgDestination.createGraphics();      AffineTransform affinetransformation = AffineTransform.getScaleInstance(2, 2);      g.drawRenderedImage(imgSource, affinetransformation);      ImageIO.write(imgDestination, "JPG", new File("outImage.jpg"));    }  } |

28. Capturing Mouse Hover Coordinates

By implementing MouseMotionListner Interface, mouse events can be captured. When the mouse is entered in a specific region MouseMoved Event is triggered and motion coordinates can be captured. The following example explains it:

|  |  |
| --- | --- |
| 01  02  03  04  05  06  07  08  09  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31 | import java.awt.event.\*;  import javax.swing.\*;    public class MouseCaptureDemo extends JFrame implements MouseMotionListener  {      public JLabel mouseHoverStatus;        public static void main(String args[])      {          new MouseCaptureDemo();      }        MouseCaptureDemo()      {          setSize(500, 500);          setTitle("Frame displaying Coordinates of Mouse Motion");            mouseHoverStatus = new JLabel("No Mouse Hover Detected.", JLabel.CENTER);          add(mouseHoverStatus);          addMouseMotionListener(this);          setVisible(true);      }        public void mouseMoved(MouseEvent e)      {          mouseHoverStatus.setText("Mouse Cursor Coordinates => X:"+e.getX()+" | Y:"+e.getY());      }        public void mouseDragged(MouseEvent e)      {}  } |

29. FileOutputStream Vs. FileWriter

File writing in Java is done mainly in two ways: FileOutputStream and FileWriter. Sometimes, developers struggle to choose one among them. This example helps them in choosing which one should be used under given requirements. First, let’s take a look at the implementation part:

Using FileOutputStream:

|  |  |
| --- | --- |
| 1  2  3  4 | File foutput = new File(file\_location\_string);  FileOutputStream fos = new FileOutputStream(foutput);  BufferedWriter output = new BufferedWriter(new OutputStreamWriter(fos));  output.write("Buffered Content"); |

Using FileWriter:

|  |  |
| --- | --- |
| 1  2  3 | FileWriter fstream = new FileWriter(file\_location\_string);  BufferedWriter output = new BufferedWriter(fstream);  output.write("Buffered Content"); |

According to Java API specifications:

*FileOutputStream is meant for writing streams of raw bytes such as image data. For writing streams of characters, consider using FileWriter.*

This makes it pretty clear that for image type of Data FileOutputStream should be used and for Text type of data FileWriter should be used.