**3. String class**

**Programmmmm to write in book**

**Remaining to execute the prg**

19. String concat(String s) or + operator

22. String[] split(String delimiter)

String[] split(String delimter,int limit)

- used to split the string based on delimiter

23. boolean matches(String regex)

- used to match the string based on regex

24. boolean regionMatches(boolean ignorecase, int toffset, String other, int offset, int length)

- used to match part of string with another string

25. int codePointAt(int index)

26. int codePointBefore(int index)

27. int codePointAfter(int index)

28. int codePointCount(int beginindex,int endindex)

29. boolean contains(String s)

30. static String join(String delimiter,String...s) - used to join string based on delimiter

31. static String copyValueOf(char[] c)

static String copyValueOf(char[] c,int offset,int count)

32. static String format(String format,Object val)

System.out.println("abcd".codePointBefore(2)); //98

System.out.println("abcdefg".codePointCount(0, 4)); //4

System.out.println(String.join("-", "one","two","three")); //one-two-three

char c[]= {'a','b','c','d','e','f','g'};

String s14="";

System.out.println(s14.copyValueOf(c)); //abcdefg

System.out.println(s14.copyValueOf(c, 3, 2)); //de

int a=23;

System.out.println(String.format("%d",a)); //23

System.out.println(String.format("|%d|", a)); //|23|

System.out.println(String.format("|%5d|", a)); //| 23| //5-total width

System.out.println(String.format("|%-5d|", a)); //|23 |

System.out.println(String.format("|%06d|",a)); //|000023|

//System.out.println(String.format("|%-06d|",a)); //exception

System.out.println(String.format("|%f|", 345.334));//|345.334000|

System.out.println(String.format("|%.2f|", 345.334)); //|345.33|

System.out.println(String.format("|%8.2f|", 345.334)); //| 345.33|

**4) StringBuffer class**

- present in java.lang.\*

- It is a mutable class, variable length of char (ie) we can increase or decrease its size at runtime

- we cant overide equals() in StringBuffer, using toString() we convert StringBuffer to String and then apply equals()

- By default StringBuffer is synchronized or threadsafe

- Default capacity of StringBuffer is 16

Constructor

1. StringBuffer()

2. StringBuffer(String s)

3. StringBuffer(int capacity)

Methods

1. int length()

2. char charAt(int position)

3. void setCharAt(int index,char c)

4. void setLength(int length)

5. int capacity()

6. StringBuffer append(int i)

StringBuffer append(char c)

StringBuffer append(String s)

- adding at end of StringBuffer

7. StringBuffer insert(int index,int n)

StringBuffer insert(int index,char c)

StringBuffer insert(int index,String s)

8. StringBuffer reverse()

9. StringBuffer replace(int start,int end, String s)

10. StringBuffer delete(int start,int end) - delete group of char

11. StringBuffer deleteCharAt(int index) - delete single char

public class Main {

public static void main(String[] args) {

StringBuffer sb1=new StringBuffer("Hello");

System.out.println(sb1); //Hello

System.out.println(sb1.length()); //5

System.out.println(sb1.capacity()); //16+5=21

System.out.println(sb1.charAt(1)); //e

sb1.setCharAt(1, 'i');

System.out.println(sb1); //Hillo

sb1.setLength(2);

System.out.println(sb1); //Hi

System.out.println(sb1.length()); //2

int b=20;

StringBuffer sb2=new StringBuffer();

String s1=sb2.append("a=").append(b).append("!").toString();

System.out.println(s1); //a=20!

StringBuffer sb3=new StringBuffer("I Java");

System.out.println(sb3.insert(2, "like ")); //I like Java

StringBuffer sb4=new StringBuffer("hello");

System.out.println(sb4.reverse()); //olleh

StringBuffer sb5=new StringBuffer("This is a test");

System.out.println(sb5.replace(5, 7, "was ")); //This was a test

StringBuffer sb6=new StringBuffer("This is a test");

System.out.println(sb6.delete(5, 7)); //This a test

System.out.println(sb6.deleteCharAt(0)); //his a test

}

}

5. StringBuilder class

- Available from JDK1.5

- Similar to StringBuffer class, it is not synchronized or thread safe so it gives better performance than ur StringBuffer class

6. Throwable class

- used to handle exception in Java

2 types of exception

1. Checked exception

- all subclasses of Exception class excluding RuntimeException. Eventhough we have written 100% correct prg also, ur code has to be surrounded by try/catch block or throws keyword otherwise ur prg will not compile.

- Will insist the programmer to surround the code by try/catch block or throws keyword otherwise ur prg will not compile

2. Unchecked Exception

- all subclassess of RuntimeException class

- Eventhough ur prg contains exception, the compiler will just compile the program but at runtime we get related exception

- Will not insist the programmer to surround the code by try/catch block or throws keyword, the prg will compile, but at runtime we get related exception

Constructor

1. Throwable()

2. Throwable(String msg)

3. Throwable(String msg,Throwable t)

Types of Exception

1. ArthimeticException - divide anything by 0

2. ArrayIndexOutOfBoundsException - datatype array

int a[]=new int[3];

a[3]=5;

3. StringIndexOutOfBoundsException - only String array

String a[]=new String[3];

a[3]="5";

4. NegativeArraySizeException

int a[]=new int[-3];

5. NumberFormatException

int a=Integer.parseInt("abc");

6. ArrayStoreException

int a[]=new int[3];

a[0]="hello";

7. NullPointerException

class A {

void show() {

}

}

class Main { PSVM {

A a=new A();

a.show();

a=null;

a.show(); //NPE

***DAY 8***

Exception Handling

5 keywords

1. try

- Program to be monitored for exception has to be put inside try block

2. catch

- used to catch the exception generated, mainly used to print userdefined messages when an exception occurs

3. finally

- optional stmt, it will be executed every time irrespective of exception occurs or not

- finally stmt used for closing resources,release memory etc

- used in 3 cases - file, database, socket programming

try {

fp=fopen("a.txt","r");

read operation

}

catch(Exception e){

}

finally {

fclose(fp);

}

3 ways

1. try { 2. try{ 3. try{

} } }

catch(Exception e){ catch(Exception e){ finally{

} } }

finally{

}

- We should not write any code between try catch and finally

Command line argument

- we give input while running the prg in command line

- When we use args argument inside the prg, then for that prg we need to give input thru command line

- each command line args should be separated by space

public class Main {

public static void main(String[] args) {

try {

int a=Integer.parseInt(args[0]);

int c=10/a;

System.out.println(c);

}

catch(ArithmeticException e) {

System.out.println("Number divided by 0: "+e);

}}}

Multi catch statement

- single try can contain multiple catch blocks

public class Main {

public static void main(String[] args) {

try {

int a=Integer.parseInt(args[0]);

int c=10/a;

System.out.println(c); //5

int b[]= {10};

b[5]=20;

}

catch(ArithmeticException e) {

System.out.println("Number divided by 0: "+e);

}

catch(ArrayIndexOutOfBoundsException e) {

System.out.println("Array Index: "+e);

}}}

If we want to invoke any other exception then we can define catch block with general class called Exception or Throwable

public class Main {

public static void main(String[] args) {

try {

int a=Integer.parseInt(args[0]);

int c=10/a;

System.out.println(c); //5

int b[]= {10};

b[5]=20;

}

catch(ArithmeticException e) {

System.out.println("Number divided by 0: "+e);

}

catch(ArrayIndexOutOfBoundsException e) {

System.out.println("Array Index: "+e);

}

catch(NullPointerException e) {

System.out.println(e);

}

catch(NumberFormatException e) {

System.out.println(e);

}

catch(Exception e) { //catch(Throwable e)

System.out.println(e);

}}}

- Whenever we define general class called Exception or Throwable, it should be always present in last catch block otherwise it leads to compilation error

public class Main {

public static void main(String[] args) {

try {

int a=Integer.parseInt(args[0]);

int c=10/a;

System.out.println(c); //5

int b[]= {10};

b[5]=20;

}

catch(Exception e) { //catch(Throwable e)

System.out.println(e);

}

catch(ArithmeticException e) { //error

System.out.println("Number divided by 0: "+e);

}

catch(ArrayIndexOutOfBoundsException e) { //error

System.out.println("Array Index: "+e);

}

catch(NullPointerException e) { //error

System.out.println(e);

}

catch(NumberFormatException e) { //error

System.out.println(e);

}}}

- From JDK1.7 version, we define multiple exception in a single catch block using | symbol

public class Main {

public static void main(String[] args) {

try {

int a=Integer.parseInt(args[0]);

int c=10/a;

System.out.println(c); //5

int b[]= {10};

b[5]=20;

}

catch(ArithmeticException | ArrayIndexOutOfBoundsException | NullPointerException | NumberFormatException e) {

System.out.println(e);

}

catch(Exception e) {

System.out.println(e);

}}}

4. throw keyword

- used to manually throw an exception

- whenever it invokes throw keyword it will automatically goes to related catch block

Syntax: throw new Exception("messages");

public class Main {

static void demo() {

try {

throw new NullPointerException("Demo");

}

catch(NullPointerException e) {

System.out.println("Caught");

throw e;

}}

public static void main(String[] args) {

try {

demo();

}

catch(NullPointerException e) {

System.out.println("Recaught");

}}}

5. throws keyword

- throws keyword used to declare an exception and used only in methods

- used to indicate that the methos might throw one of the exception

public class Main {

static void demo() throws NullPointerException {

throw new NullPointerException("Demo");

}

public static void main(String[] args) {

try {

demo();

}

catch(NullPointerException e) {

System.out.println("Recaught");

}}}

public class Main {

static void demoA() {

try {

System.out.println("Inside demoA");

throw new RuntimeException("Hello");

}

finally {

System.out.println("Inside demoA finally");

}

}

static void demoB() {

try { System.out.println("Inside demoB");

return;

}

finally {

System.out.println("DemoB finally");

}

}

public static void main(String[] args) {

try {

demoA();

}

catch(RuntimeException e) {

System.out.println("caught");

}

demoB();

}

}

Userdefined Exceptions

- Your userdefined exception class should extend Exception class and override toString()

class NotValidAgeException extends Exception {

String s1="";

public NotValidAgeException(String s1) {

super();

this.s1 = s1;

}

@Override

public String toString() {

return s1;

}}

public class Main {

static void validateAge() throws NotValidAgeException {

Scanner sc=new Scanner(System.in);

System.out.println("Enter age");

int age=sc.nextInt();

if(age<18) {

throw new NotValidAgeException("Your age is not eligible");

}

else {

System.out.println("You are eligible");

}

}

public static void main(String[] args) {

try {

validateAge();

} catch (NotValidAgeException e) {

System.out.println(e);

}}}

Assertions

- Available from JDK1.5

- Used to validate the boolean conditions at runtime

Syntax: assert <<expression>>;

assert <<expression>>:String message;

- By default assertions is disabled in Java, while running we have to enable assertion using -ea

- Provided with assert keyword and AssertionError class is unchecked exception because it inherits from Error class

- If assert condition fails it throws AssertionError class

public class Main {

static double withdraw(double balance, double amount) {

assert (balance>=amount):"Balance is insufficient";

return (balance-amount);

}

public static void main(String[] args) {

System.out.println(withdraw(1000,500)); //500.0

System.out.println(withdraw(1000,2000)); //-1000.0

}}

try with resources

- Available from JDK1.7

- Used to close automatically all the resources at the end of stmt, using AutoCloseable interface

Syntax:

try(resources) { }

catch(Exception e){ }

public class Main {

public static void main(String[] args) {

String line;

try (BufferedReader br=new BufferedReader(new FileReader("Main.java"));

PrintWriter pw=new PrintWriter(new File("a.txt"))){

while((line=br.readLine()) != null) {

System.out.println(line);

}}

catch(Exception e) {

System.out.println(e);

}}}

In JDK1.9, enhancement in try with resouces, we can define resources outside the try and use it

public class Main {

public static void main(String[] args) throws IOException {

String line;

BufferedReader br=new BufferedReader(new FileReader("Main.java"));

PrintWriter pw=new PrintWriter(new File("a.txt"));

try (br;pw){

while((line=br.readLine()) != null) {

System.out.println(line);

}}

catch(Exception e) {

System.out.println(e);

}}}

java.util.\*

- Utility framework or Collection framework

- used to store collection of objects

1. Collection interface - core interface in order to store of collection of objects

2. Collections class - provided with static algorithm/methods that supports util package

Collection interface

- core interface in order to store of collection of objects

Methods

1. boolean add(Object obj) - add single object

2. boolean addAll(Collection c) - add multiple objects

3. boolean remove(Object obj) - remove single object

4. boolean removeAll(Collection obj) - remove all object

5. boolean contains(Object obj) - check single object present in collection or not

6. boolean containsAll(Collection c) - check multiple object present in collection or not

7. boolean retainAll(Collection c) - remove from the target collection all the elements that are not contained in the specified collection

8. int size() - return number of object present in collection

9. Object[] toArray() - return an array containing all elts in collection

10. Iterator iterator()

11. ListIterator listIterator()

12. boolean isEmpty()

13. boolean equals(Object o)

List interface

- Ordered and duplicate elements

Methods

1. void add(int index,Object obj) - add single object at particular index position

2. boolean addAll(int index,Collection c) - add multiple object at particular index position

3. Object get(int index) - return single object at particular index position

4. int indexOf(Object o) - return position of first occurence of object in collection

5. int lastIndexOf(Object o) - return position of last occurence of object in collection

6. Object remove(int index) - remove single object present in particular index

7. Object set(int index, Object o) - set is something like replace

8. List subList(int start,int end) - return part of list from start to end-1

Set interface

- unordered and no duplicates elt

SortedSet interface

- used to sort the elements in Set interface

Methods

1. Object first() - return first object

2. Object last() - return last object

3. SortedSet subSet(Object start,Object end) - part of set from start to end-1

4. SortedSet headSet(Object o) - return all elts present before specified elt

1 2 3 4 5.headSet(3); //1 2

5. SortedSet tailSet(Object o) - return all elts present after specified elt

1 2 3 4 5.tailSet(3); //4 5