DAY -5

Day-4:Strings,Wrapper classes, Immutability, Inner classes,Lambda expression, Java 5 features , IO, Exception Handling

- String class: Immutability and usages, stringbuilder and stringbuffer
- Wrapper classes and usages, Java 5 Language Features
- AutoBoxing and Unboxing, Enhanced For loop, Varargs, Static Import, Enums
- Inner classes: Regular inner class, method local inner class, annonymous inner class
- Char and byte oriented streams
- BufferedReader and BufferedWriter
- File handling
- Object Serialization and IO [ObjectInputStream / ObjectOutputStream]
- Exception Handling, Types of Exception, Exception Hierarchy, Exception wrapping and re throwing
- Hands On & Lab

String

- Immutable i.e. once assigned then can't be changed
- Only class in java for which object can be created with or without using new operator

Ex: String s="india";
String s1=new String("india");
What is the difference?

- String concatenation can be in two ways:
  - String s1=s+ "paki"; Operator overloading
  - String s3=s1.concat("paki");

rgupta.mtech@gmail.com

Immutability

- Immutability means something that cannot be changed.
- Strings are immutable in Java. What does this mean?
  - String literals are very heavily used in applications and they also occupy a lot of memory.
  - Therefore for efficient memory management, all the strings are created and kept by the JVM in a place called string pool (which is part of Method Area).
  - Garbage collector does not come into string pool. How does this save memory?

RULES FOR IMMUTABILITY

Declare the class as final so it can't be extended.

Make all fields private so that direct access is not allowed.

Don't provide setter methods for variables

Make all mutable fields final so that it's value can be assigned only once.

Initialize all the fields via a constructor performing deep copy.

Perform cloning of objects in the getter methods to return a copy rather than returning the actual object reference.

rgupta.mtech@gmail.com

How to make copy of an Object?

```
int[] src  = new int[]{1,2,3,4,5};
int[] dest = new int[5];
System.arraycopy( src, 0, dest, 0, src.length );
```

```
int[] a = new int[]{1,2,3,4,5};
int[] b = a.clone();
```

```
int[] a = {1,2,3,4,5};
int[] b = Arrays.copyOf(a, a.length);
```

Arrays.copyOfRange():
If you want few elements of an array to be copied

```
public final class ImmutableReminder{
    private final Date remindingDate;

    public ImmutableReminder (Date remindingDate) {
        if(remindingDate.getTime() < System.currentTimeMillis()){
            throw new IllegalArgumentException("Can not set reminder"
                        " for past time: " + remindingDate);
        }
        this.remindingDate = new Date(remindingDate.getTime());
    }

    public Date getRemindingDate() {
        return (Date) remindingDate.clone();
    }
}
```

rgupta.mtech@gmail.com

Some common string methods…

charAt()
Returns the character located at the specified index
concat()
Appends one String to the end of another ("+" also works)

Determines the equality of two Strings, ignoring case
length()
Returns the number of characters in a String
replace()
Replaces occurrences of a character with a new character
substring()
Returns a part of a String
toLowerCase()
Returns a String with uppercase characters converted
toString()
Returns the value of a String
toUpperCase()
Returns a String with lowercase characters converted
equalsIgnoreCase()
Removes whitespace from the ends of a String

rgupta.mtech@gmail.com

Some common string methods...

String	methods
charAt()	Returns the character located at the specified index
concat()	Appends one String to the end of another ("+" also works)
equalsIgnoreCase()	Determines the equality of two Strings, ignoring case
length()	Returns the number of characters in a String
replace()	Replaces occurrences of a character with a new character
substring()	Returns a part of a String
toLowerCase()	Returns a String with uppercase characters converted
toString()	Returns the value of a String
toUpperCase()	Returns a String with lowercase characters converted
trim()	Removes whitespace from the ends of a String

rgupta.mtech@gmail.com

String comparison

- Two string should never be checked for equality using == operator
  WHY?
- Always use equals( ) method....

String s1="india"; String s2="paki";

```
if(s1.equals(s2))
      .....
      .....
```

rgupta.mtech@gmail.com

	String	StringBuffer	StringBuilder
Storage Area	Constant String Pool	Heap	Heap
Modifiable	No (immutable)	Yes(mutable)	Yes(mutable)
Thread Safe	Yes	Yes	No
Performance	Fast	Very slow	Fast

Various memory area of JVM

1. **Method area ------ Per JVM**
2. **heap area ------ Per JVM**
3. **stack area ------ Per thread**
4. **PC register area ------ Per thread**
5. **Native method area ------ Per thread**

String	StringBuffer / StringBuilder (added in J2SE 5.0)
String is **immutable**: you can't modify a string object but can replace it by creating a new instance. Creating a new instance is rather expensive.	StringBuffer is **mutable**: use StringBuffer or StringBuilder when you want to modify the contents. ***StringBuilder*** was added in Java 5 and it is identical in all respects to ***StringBuffer*** except that it is not synchronized, which makes it slightly faster at the cost of not being thread-safe.

//Inefficient version using immutable String
String output = "Some text"
Int count = 100;
for(int i =0; i<count; i++) {
output += i;
}
return output;
	//More efficient version using mutable StringBuffer
	StringBuffer output = new **StringBuffer(110)**;// set an initial size of 110
	output.append("Some text");
	for(int i =0; i<count; i++) {
	output.append(i);
	}
	return output.toString();

The above code would build 99 new String objects, of which 98 would be thrown away immediately. Creating new objects is not efficient.	The above code creates only two new objects, the *StringBuffer* and the final *String* that is returned. StringBuffer expands as needed, which is costly however, so it would be better to initialize the *StringBuffer* with the correct size from the start as shown.

Wrapper classes

Helps treating primitive data as an object
But why we should convert primitive to objects?

- We can't store primitive in java collections
- Object have properties and methods
- Have different behavior when passing as method argument

Eight wrapper for eight primitive

Integer, Float, Double, Character, Boolean etc…

```
Integer it=new Integer(33);  int temp=it.intValue();
....
```

```
primitive==>object
Integer it=new Integer(i);

object==>primitive
int i=it.intValue()

primitive ==>string
String s=Integer.toString();

String==>Numeric object
Double val=Double.valueOf(str)
```

rgupta.mtech@gmail.com

Boxing / Unboxing Java 1.5

- **Boxing**

  Integer iWrapper = 10; Prior to J2SE 5.0, we use
  Integer a = new Integer(10);

- **Unboxing**

  int iPrimitive = iWrapper;
  Prior to J2SE 5.0, we use
  int b = iWrapper.intValue();

rgupta.mtech@gmail.com

Java 5 Language Features (I)

AutoBoxing and Unboxing

- Enhanced For loop
- Varargs

Covariant Return Types

Static Import

Enums

rgupta.mtech@gmail.com

Enhanced For loop

Provide easy looping construct to loop through array and collections

```
int x[]=new int[5];
....
....
for(int temp:x)
        Sysout("temp:"+temp);

class Animal{ }
class Cat extends Animal{   }
class Dog extends Animal{  }

Animal []aa={new Cat(),new Dog(), new Cat()};

for(Animal a:aa)
        .......
```

rgupta.mtech@gmail.com

Varargs

- Java start supporting variable argument method in Java 1.5

```
class Foo{

        public void foo(int ...j)
        {
                for(int temp:j)
                        Sysout(temp);
        }
        .....
        .....
}
```

Discuss function overloading in case of Varargs and wrapper classes

rgupta.mtech@gmail.com

Static Import

- Handy feature in Java 1.5 that allow something like this: import static java.lang.Math.PI;
- import static java.lang.Math.cos;
- Now rather then using
  - ***double r = Math.cos(Math.PI * theta);***
- We can use something like ***double r = cos(PI * theta);*** – looks more readable ….
- Avoid abusing static import like
  - *import static java.lang.Math.*;*
- General guidelines to use static java import:
  - Use it to declare local copies of java constants
  - When you require frequent access to static members from one or two java classes

rgupta.mtech@gmail.com

Enums

- Enum is a special type of classes.
- enum type used to put restriction on the instance values

```
public enum myCars{
    HONDA,BMW,TOYOTA
};
```

its optional !!!

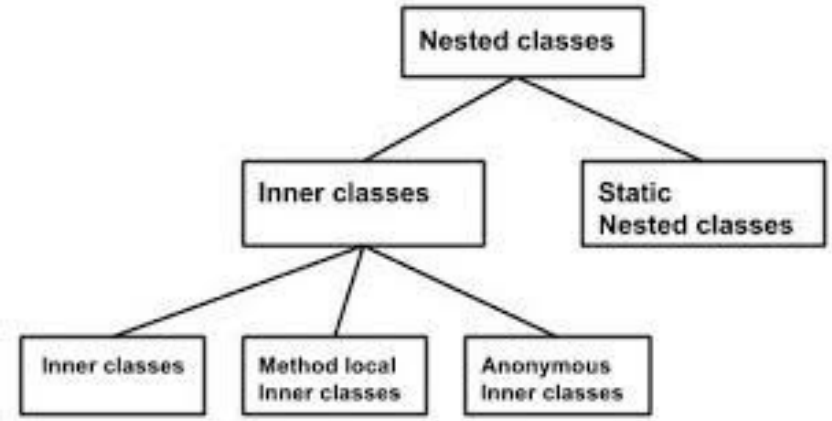
```
        ........
        ........
    myCars currentcar=myCars.BMW;
     System.out.println("My Current Cars : "+ myCars.BMW);
```

```
enum Day {
        SUNDAY, MONDAY, TUESDAY,WEDNESDAY, THURSDAY,
        FRIDAY, SATURDAY
    }

//used
Day today = Day.WEDNESDAY;
switch(today){
        case SUNDAY:
                break;
        //...
}
```

rgupta.mtech@gmail.com

Inner classes



- Inner class?
  - A class defined inside another class.
  - Why to define inner classes
- Inner classes
  - Top level inner class
  - Method local inner class
  - Anonymous Inner class
  - Method local argument inner class
- Static inner classes

rgupta.mtech@gmail.com

Top level inner class

- Non static inner class object can't be created without outer class instance
- All the private data of outer class is available to inner class
- Non static inner class can't have static members

http://www.avajava.com/tutorials/lessons/iterator-pattern.html

```
class A
{
    private int i=90;

    class B
    {
        int i=90;//
        void foo()
        {
            System.out.println("instance value of outer class:"+ A.this.i);

            System.out.println("instance value of inner class:"+this.i);
        }
    }
}
public class Inner1 {

    public static void main(String[] args) {

        A.B objectB=new A().new B();

        objectB.foo();
```

rgupta.mtech@gmail.com

Method local inner class

- Class define inside an method
- Can not access local data define inside method
- Declare local data final to access it

```
class A
{
    private int i=90;
    void foo()
    {
        int i=22;
        final int j=44;
        class B
        {
            void foofy()
            {
                //System.out.println("value of method local variable cant be access:"+i);
                System.out.println("value of method local" +
                        " can be accessed if it is declared final:"+ j);
            }
        }

        B b=new B();
        b.foofy();
    }

}
```

rgupta.mtech@gmail.com

Anonymous Inner class

- A way to implement polymorphism
- “On the fly”

```
interface Cookable
{
    public void cook();
}

class Food
{
    Cookable c=new Cookable() {
        @Override
        public void cook() {
            System.out.println("Cooked......");
        }
    };
}
```

rgupta.mtech@gmail.com

What is Exception?

- An exception is an abnormal condition that arises while running a program.

- Examples:
  - Attempt to divide an integer by zero causes an exception to be thrown at run time.
  - Attempt to call a method using a reference that is null. Attempting to open a nonexistent file for reading.
  - JVM running out of memory.

  - Exception handling do not correct abnormal condition rather it make our program robust i.e. make us enable to take remedial action when exception occurs…Help in recovering…

rgupta.mtech@gmail.com

Type of exceptions

1. **Unchecked Exception**
   Also called Runtime Exceptions
2. **Checked Exception**
   Also called Compile time Exceptions
3. **Error**
   Should not to be handled by programmer..like JVM crash

- All exceptions happens at run time in programming and also in real life.....
- for checked ex, we need to tell java we know about those problems for example readLine() throws IOException

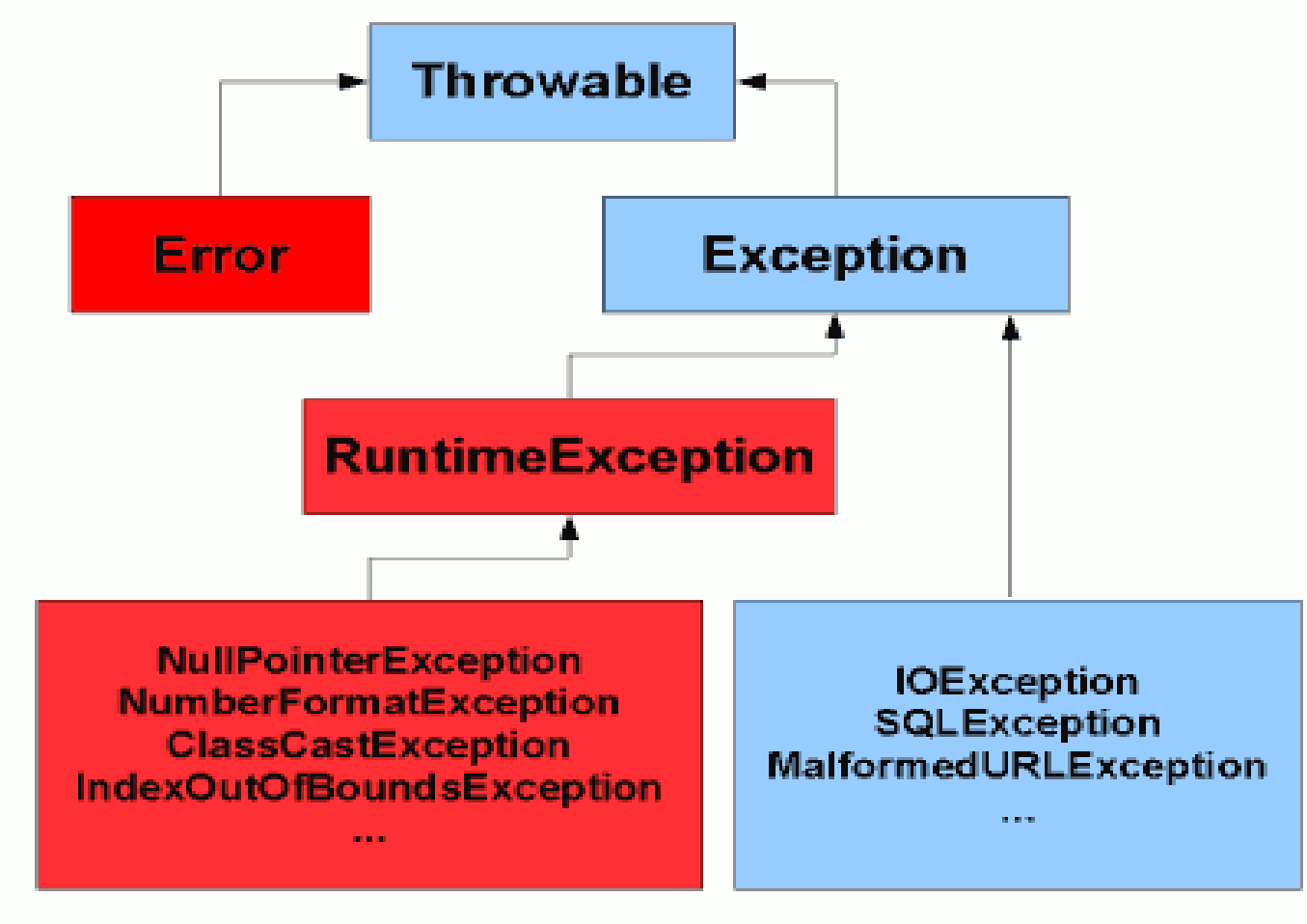
Exception key words

Catch

Throw

rgupta.mtech@gmail.com

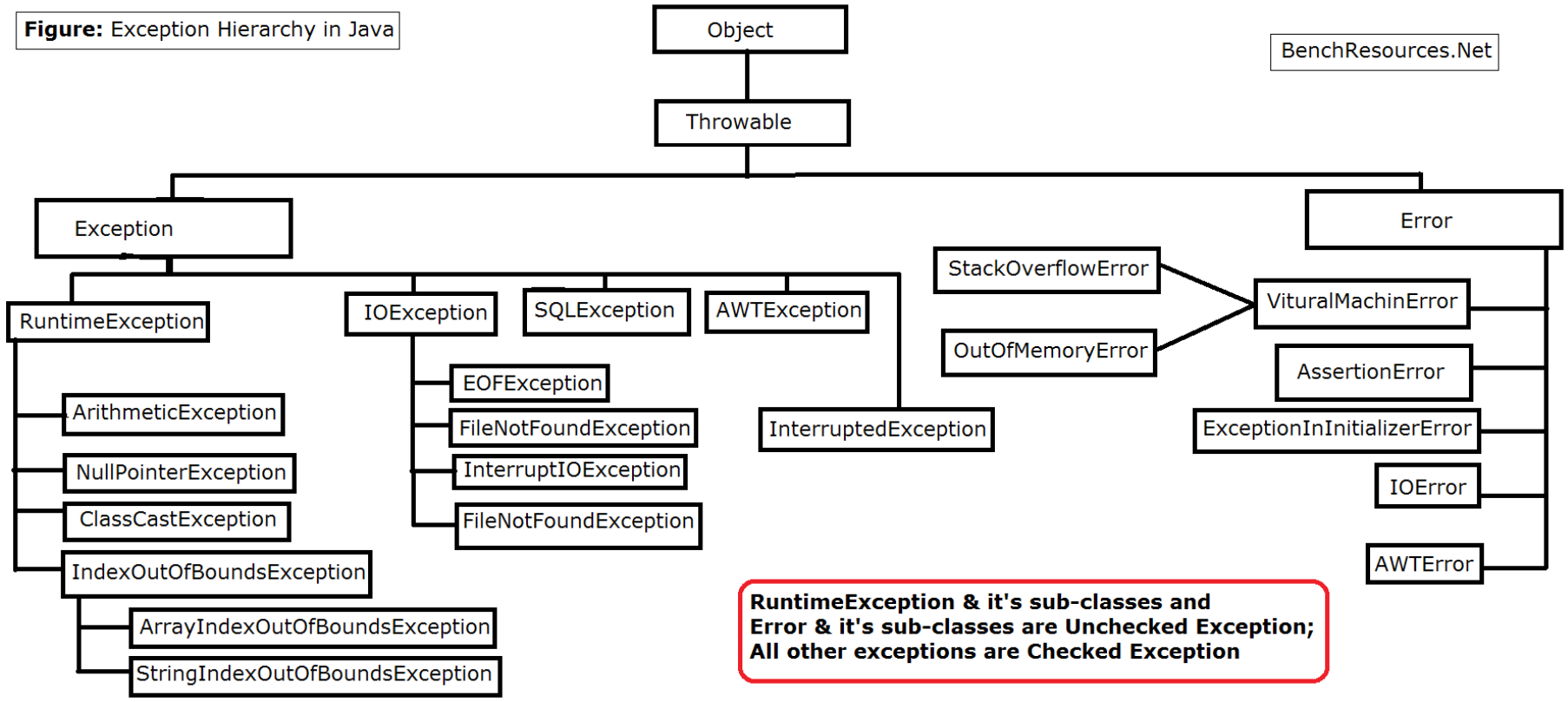
Exception Hierarchy



rgupta.mtech@gmail.com

Exception Hierarchy

Figure: Exception Hierarchy in Java



Exceptions and their Handling

When the JVM encounters an error such as divide by zero, it creates an exception object and throws it – as a notification that an error has occurred.

If the exception object is not caught and handled properly, the interpreter will display an error and terminate the program.

If we want the program to continue with execution of the remaining code, then we should try to catch the exception object thrown by the error condition and then take appropriate corrective actions. This task is known as *exception handling*.

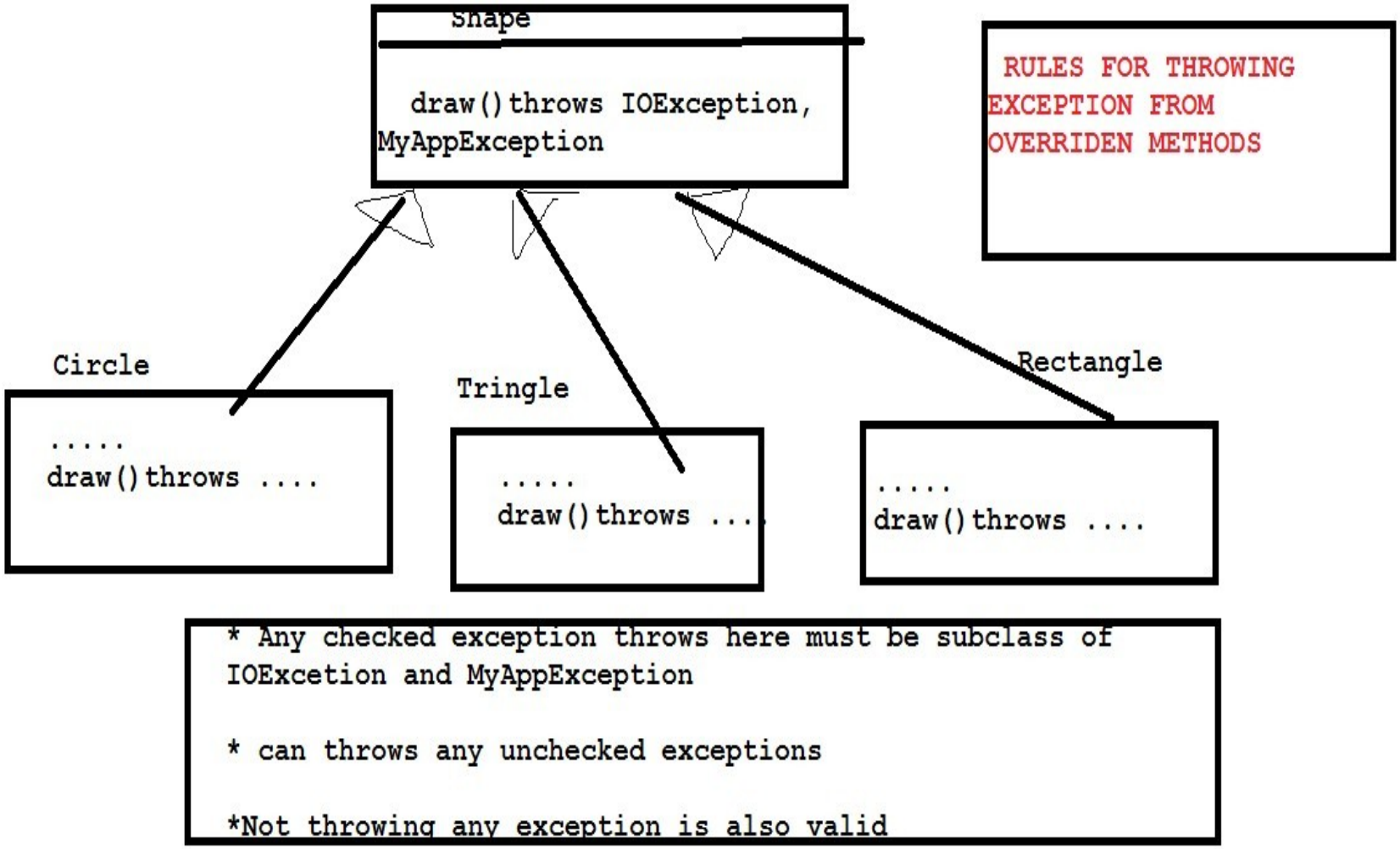
rgupta.mtech@gmail.com

Common Java Exceptions

- ArithmeticException
- ArrayIndexOutOfBoundException
- ArrayStoreException
- FileNotFoundException
- IOException – general I/O failure
- NullPointerException – referencing a null object
- OutOfMemoryError
- SecurityException – when applet tries to perform an action not allowed by the browser's security setting.
- StackOverflowError
- StringIndexOutOfBoundException

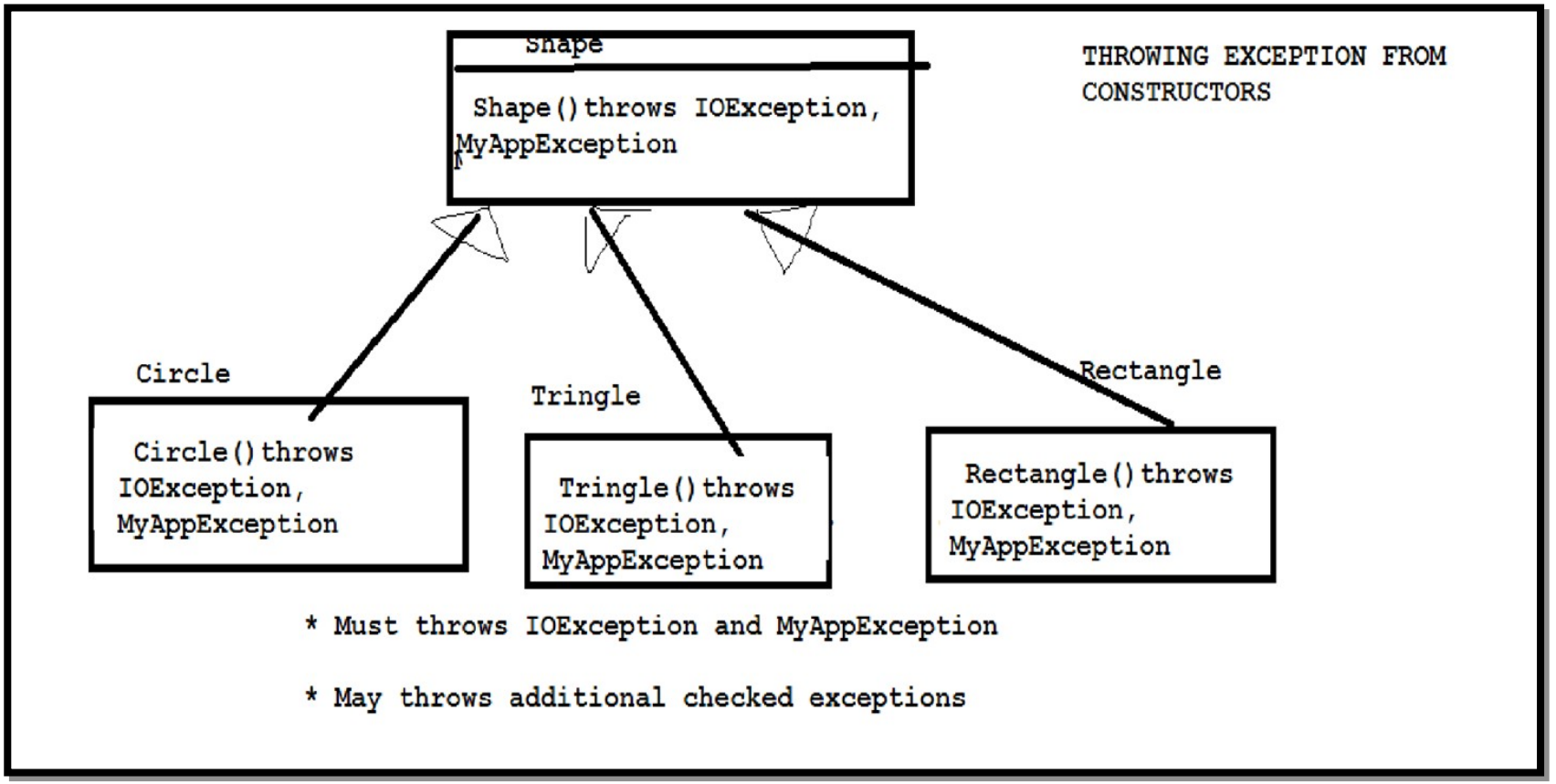
rgupta.mtech@gmail.com

Rules for throwing exception from overridden methods



rgupta.mtech@gmail.com

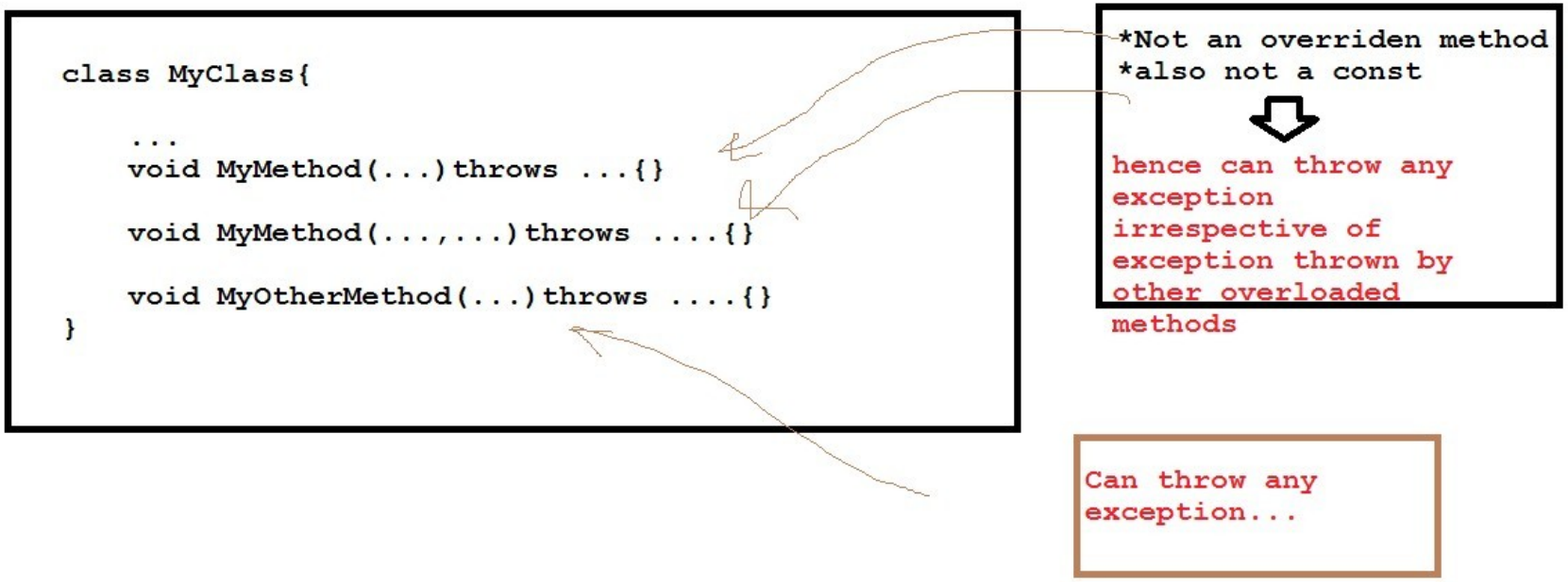
Rules for throwing exception from constructors



rgupta.mtech@gmail.com

Rules for throwing exception from overloaded and other methods

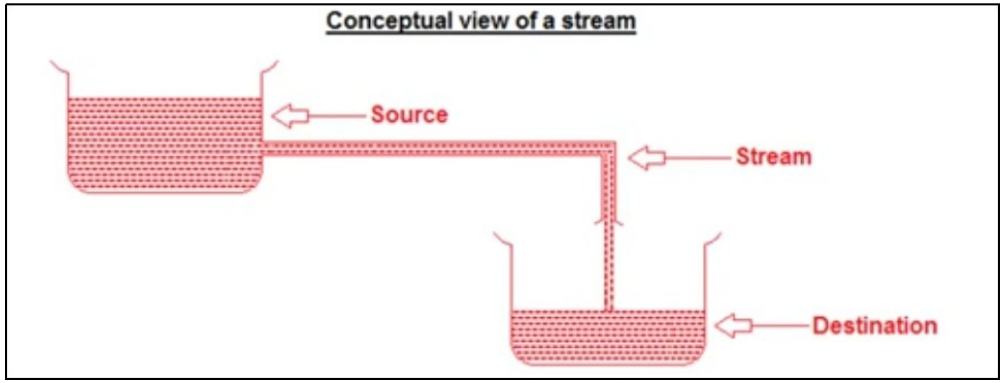
Throwing Exception from Overloaded and other methods...



rgupta.mtech@gmail.com

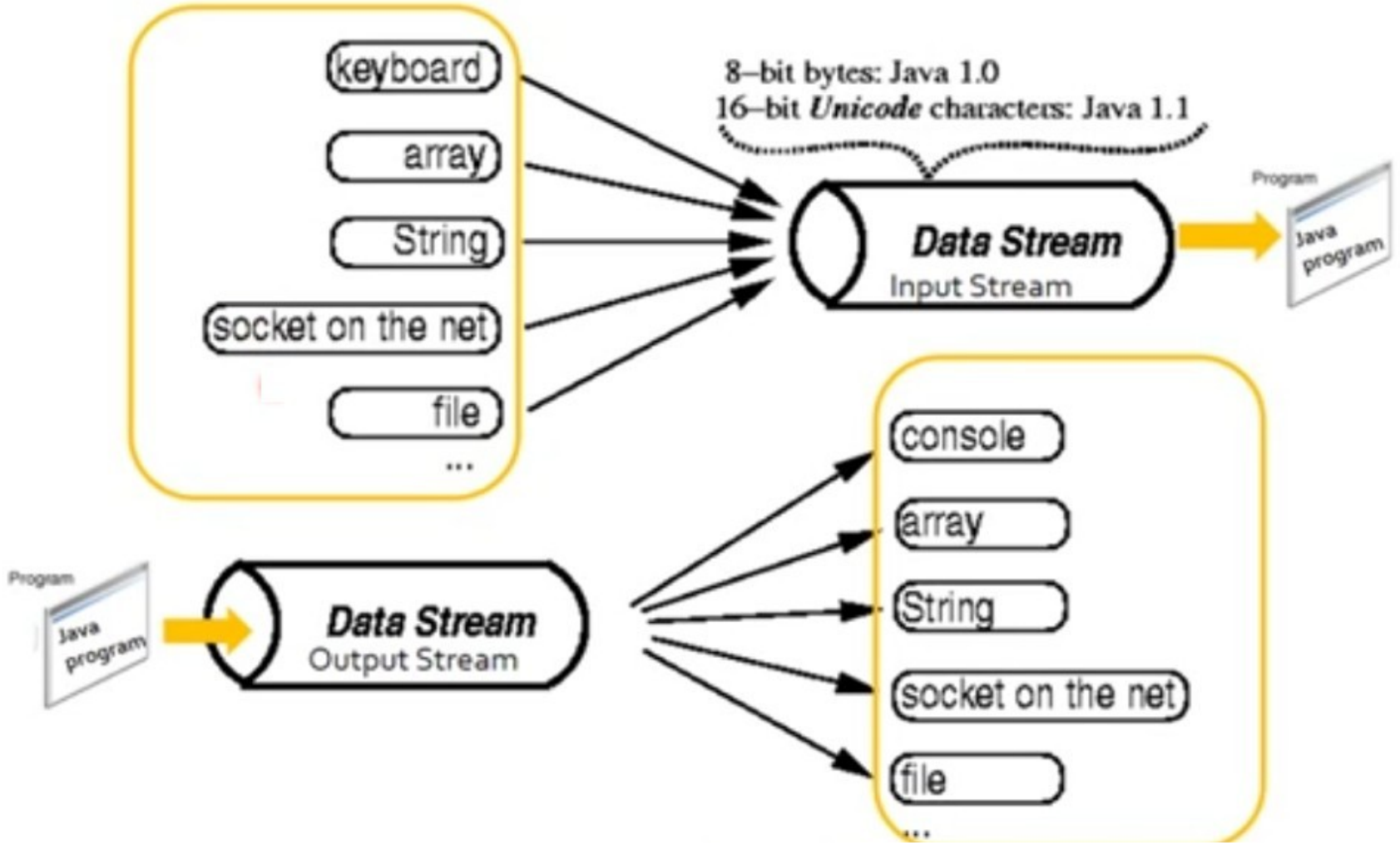
Stream and IO

- Stream is logical connection bw source and destination
- Stream is an abstraction that either produces or consumes information. A stream is linked to a physical device by the Java I/O system.
- All streams behave in the same manner, even if the actual physical devices to which they are linked differ.
- **2 types of streams:**
  - byte : for binray data, contain 0,1 sequence
  - All byte stream class are like XXXXXXXXStream character: for **text data**
  - All char stream class are like XXXXXXXXReader/ XXXXXXWriter

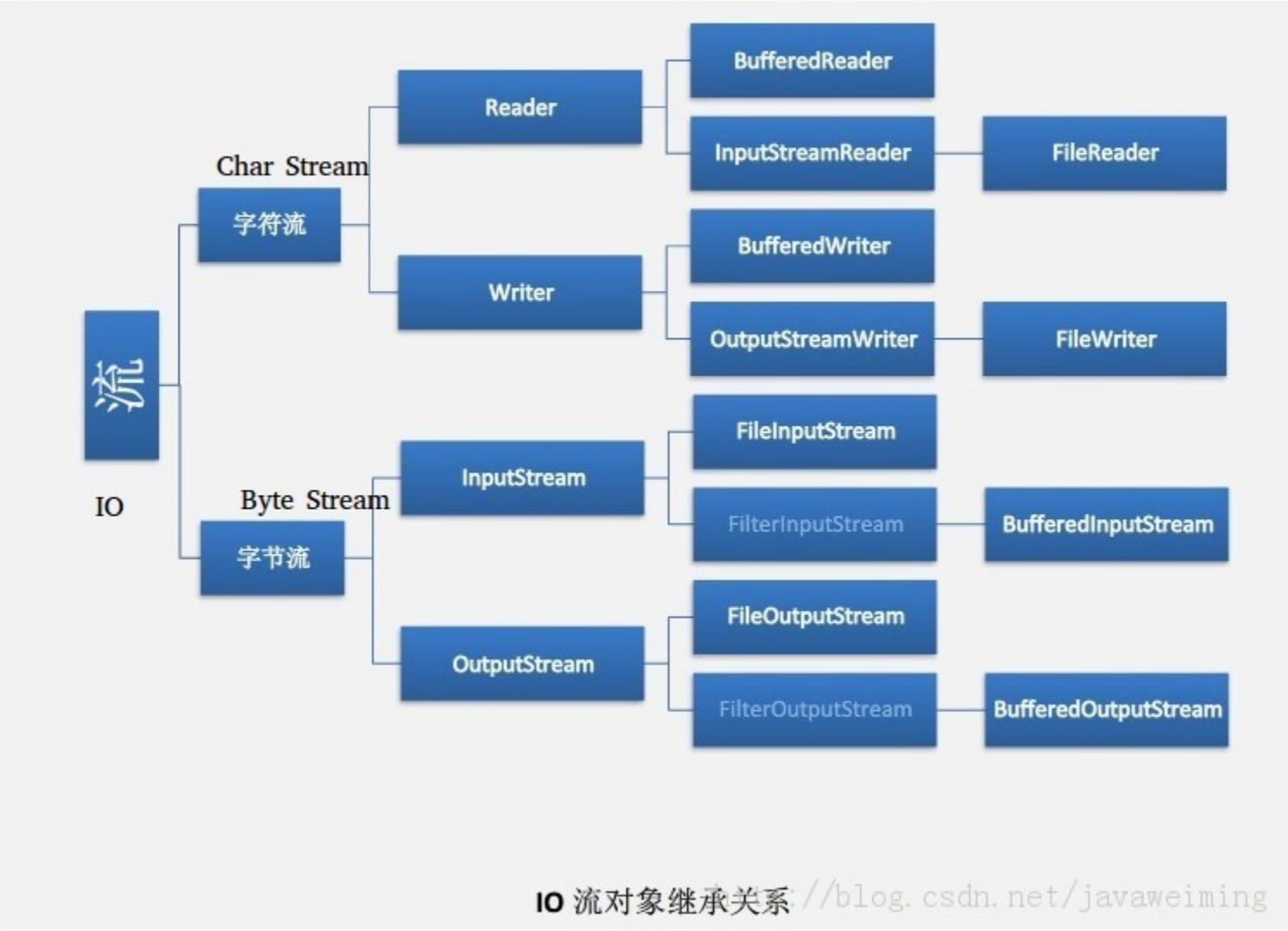


rgupta.mtech@gmail.com

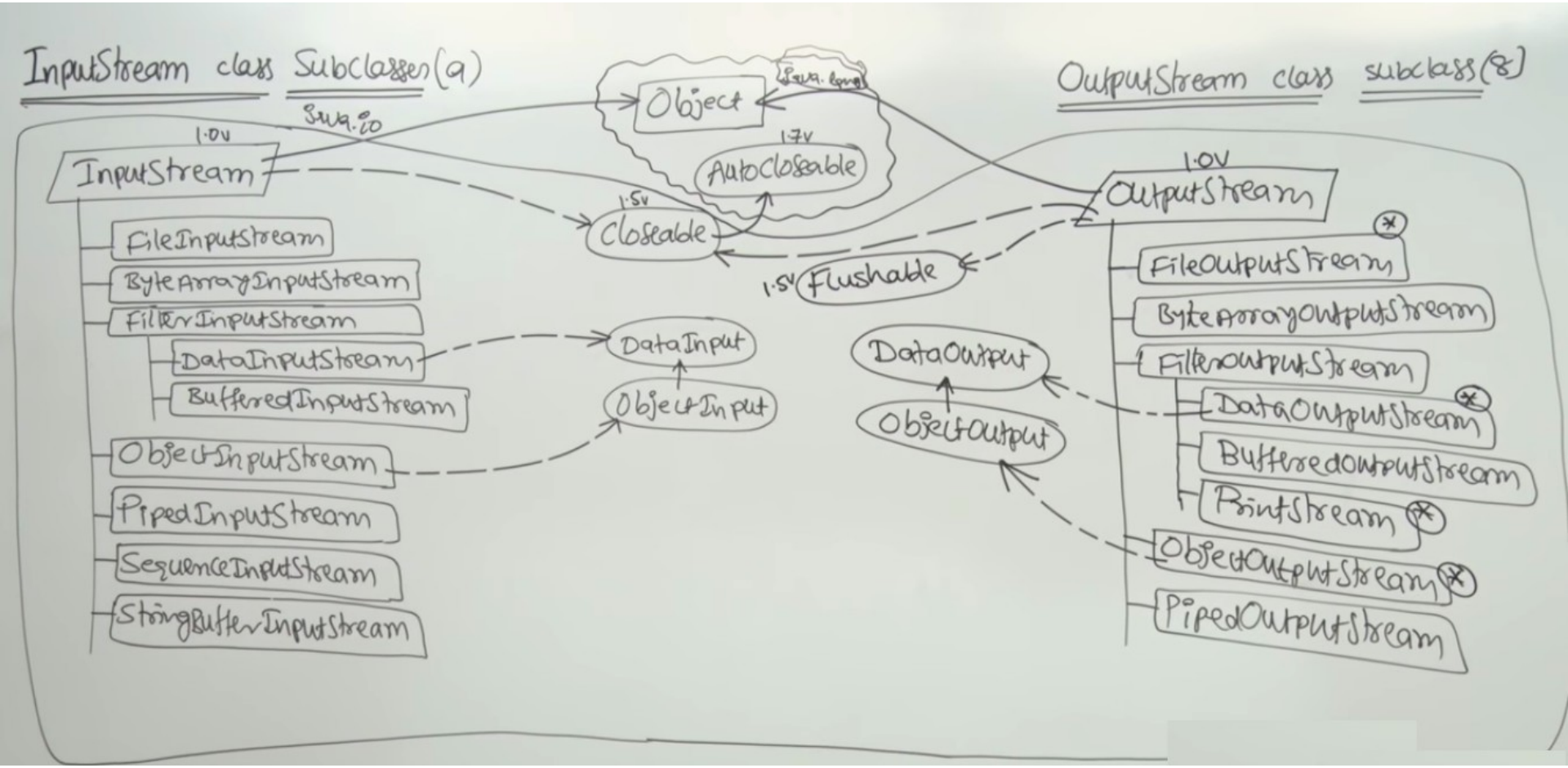
Stream and IO



Stream and IO



Stream and IO



System class in java

- System class defined in java.lang package
- It encapsulate many aspect of JRE

- System class also contain 3 predefine stream variables
  in
  - System.in (InputStream)

  out
  - System.out(PrintStream)

  err
  - System.err(console)

rgupta.mtech@gmail.com

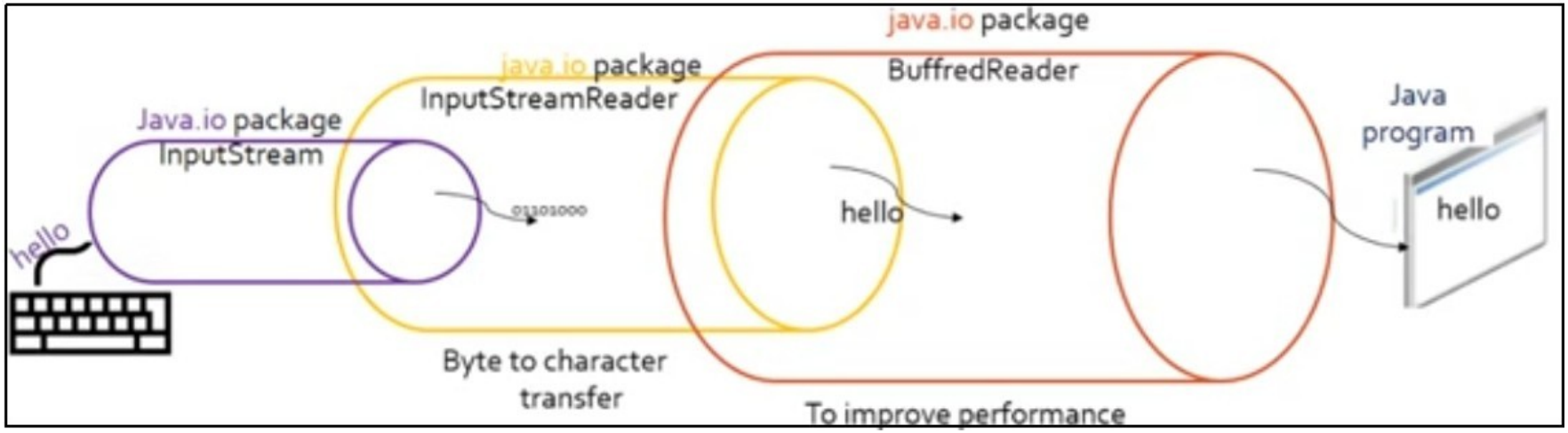
Buffered Reader and BufferedWriter

Reading form console

BufferedReader br=new BufferedReader(new InputStreamReader(System.in));

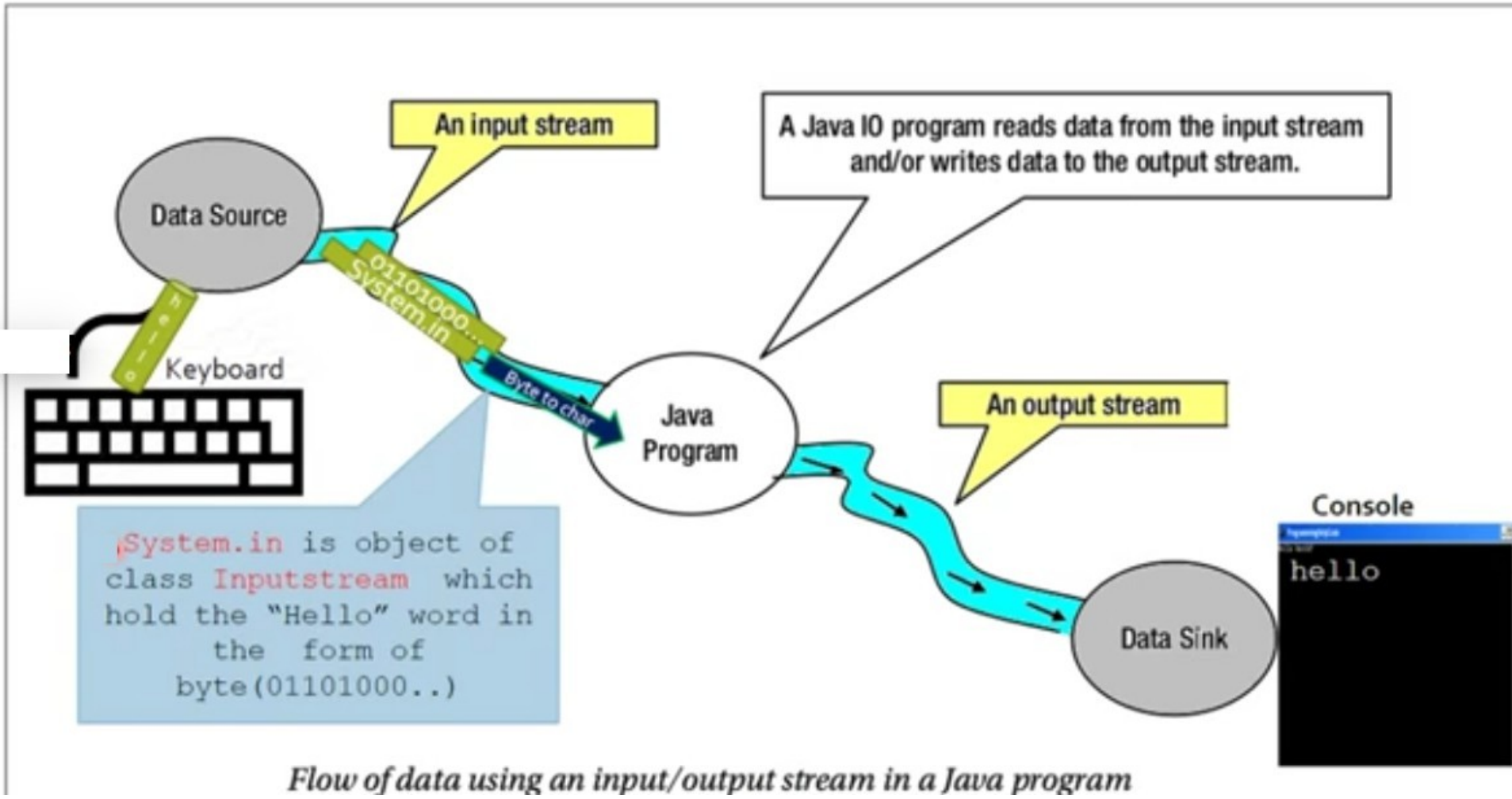
Reading form file

BufferedReader br=**new BufferedReader(new FileReader(new File("c:\\raj\\foo.txt")));**



rgupta.mtech@gmail.com

Stream as flow of data



Flow of data using an input/output stream in a Java program

File: using file, creating directories

File abstraction that represent file and directories

- File f=new File("....");
  boolean flag= file.createNewFile();
  boolean flag= file.mkDir(); boolean flag=file.exists();

```
public class DemoFileWriter {

    public static void main(String[] args) {
        try {
            BufferedWriter bw=new BufferedWriter
                    (new FileWriter("demo5.txt"));

            bw.write("java is key");
            bw.flush();
        } catch (IOException e) {
            e.printStackTrace();
        }

    }

}
```

rgupta.mtech@gmail.com

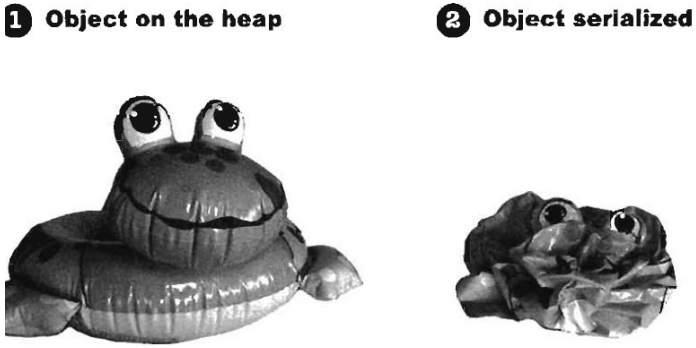
Using data output steam and data input stream

```
try {
    //writing in file
    DataOutputStream dos = new DataOutputStream
            (new FileOutputStream(
            new File("foo.txt")));
    for(int i=0; i<10;i++){
        dos.writeInt(i);
        dos.writeDouble(6.6);
        dos.writeShort(i);
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (Exception ex) {
    ex.printStackTrace();
}
```

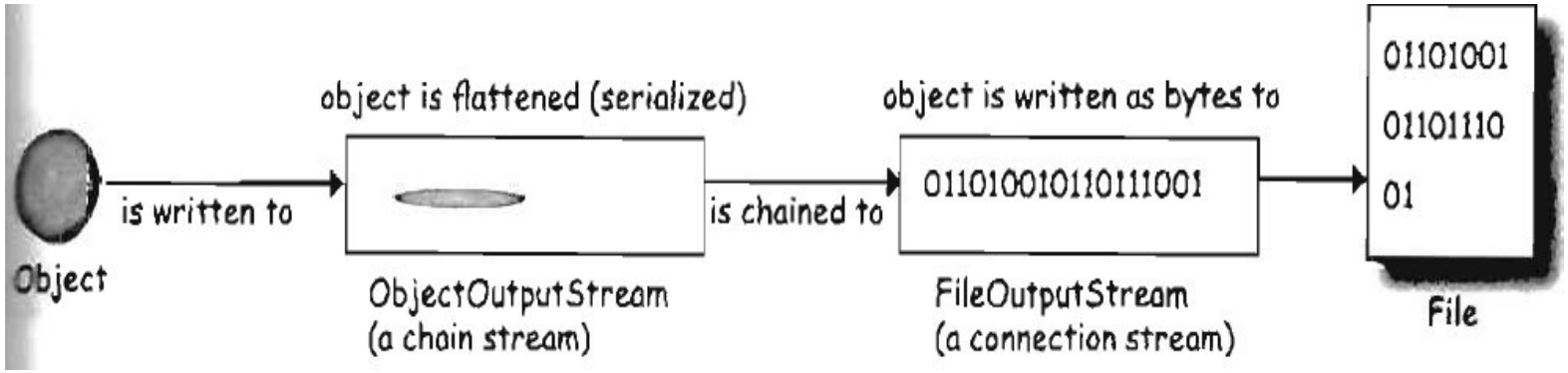
```
try {
    //reading from file
    DataInputStream dis = new DataInputStream
            (new FileInputStream(
            new File("foo.txt")));
    for(int i=0; i<10;i++){
        System.out.println(dis.readFloat());
        System.out.println(dis.readInt());
    }
} catch (FileNotFoundException e) {
    e.printStackTrace();
} catch (Exception ex) {
    ex.printStackTrace();
}
```

rgupta.mtech@gmail.com

Serialization



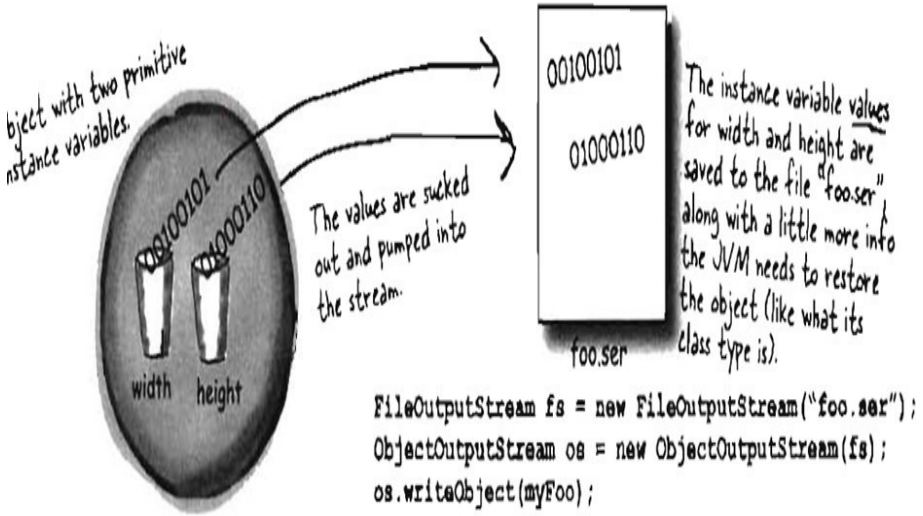
- Storing the state of the object on a file along some metadata….so that it Can be recovered back……..
- Serialization used in RMI (Remote method invocation) while sending an object from one place to another in network…



rgupta.mtech@gmail.com

What actually happens during Serialization

- The state of the object from heap is sucked and written along with some meta data in an file….

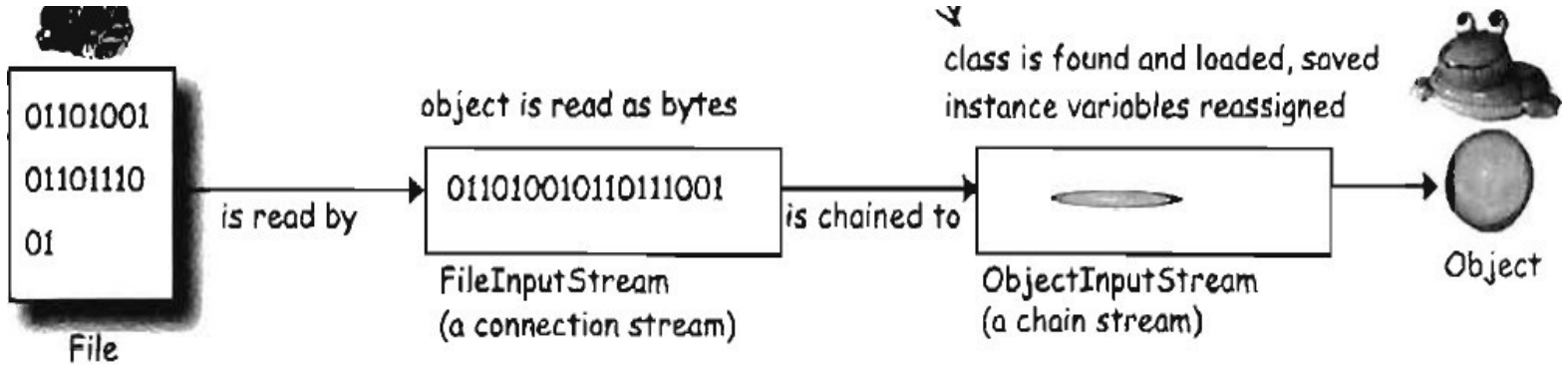


When an object is serialized, all the objects it refers to from instance variables are *also* serialized. And all the objects *those* objects refer to are serialized. And all the objects *those* objects refer to are serialized... and the best part is, it happens automatically!

rgupta.mtech@gmail.com

De- Serialization

- When an object is de-serialized, the JVM attempts to bring object back to the life by making an new object on the heap that have the same state as original object
- Transient variable don’t get saved during serialization hence come with null !!!



rgupta.mtech@gmail.com

Hello world Example...

```
class Box implements Serializable{
	private static final long serialVersionUID = 1L;
	int l,b;

	public Box(int l, int b) {
		super();
		this.l = l;
		this.b = b;
	}

	public String toString() {
		return "Box [l=" + l + ", b=" + b + "]";
	}

}
```

```
Box box=new Box(22, 33);
```

```
FileOutputStream fo=new FileOutputStream("c:\\raj\\foofoo.ser");
ObjectOutputStream os=new ObjectOutputStream(fo);

os.writeObject(box);
```

```
box=null;//nullify to prove that object come by de-ser...
```

```
FileInputStream fi=new FileInputStream("c:\\raj\\foofoo.ser");
ObjectInputStream oi=new ObjectInputStream(fi);

box=(Box)oi.readObject();
```

```
System.out.println(box);
```

rgupta.mtech@gmail.com

Differences between Serialization and Externalization?

Serialization	Externalization
It is Meant for Default Serialization.	It is Meant for Customized Serialization.
Here Everything Takes Care by JVM and Programmer doesn't have any Control.	Here Everything Takes Care by Programmer and JVM doesn't have any Control.
In Serialization Total Object will be Saved to the File Always whether it is required OR Not.	In Externalization Based on Our Requirement we can Save Either Total Object OR Part of the Object.
Relatively Performance is Low.	Relatively Performance is High.
Serialization is the best choice if we want to save total object to the file.	Externalization is the best choice if we want to save part of the Object to the file.
Serializable Interface doesn't contain any Method. It is a *Marker* Interface.	Externalizable Interface contains 2 Methods, *wrteExternal*() and *readExternal()*. So it is Not Marker Interface.
Serializable implemented Class Not required to contain public No - Argument Constructor.	Externalizable implemented Class should Compulsory contain public No - Argument Constructor. Otherwise we will get Runtime Exception Saying *InvalidClassException*.
transient Key Word will Play Role in Serialization.	transient Key Word won't Play any Role in Externalization. Of Course it is Not Required.