Question

Regex

What would be the regex to match a number that is at least 10 but not more than 9999?

`\d{2,4}`

What is the regex for an input that has seven digits and that can have + or - at the start?

`[+-]?\d{7}`

An e-mail address is
- A sequence of letters, followed by
- the character "@", followed by
- the character "." , followed by a nonempty sequence of lowercase letters, followed by
- [any number of occurrences of the previous pattern]
- "edu" or "com" (others omitted for brevity).

Give a generalized RE for e-mail addresses

`[A-Za-z]+@([a-z]\.)+(edu|com)`

Which of the following strings match the RE `a*bb(ab|ba)*`?

A. abb
B. aaba
C.abba
D.bbbaab
E.cbb
F.bbababbab

好像有点问题？？

环形数组队列

```
1  class CicleArrayQueue {
2      private int maxSize;
3      private int front;
4      private int rear;
5      private int[] arr;
```

```
6
7       public CicleArrayQueue(int maxSize) {
8           this.maxSize = maxSize+1;
9           arr = new int[maxSize + 1];
10      }
11
12      //判断队列是否满
13      public boolean isFull() {
14          return (rear + 1) % maxSize == front;
15      }
16
17      //判断队列是否为空
18      public boolean isEmpty() {
19          return rear == front;
20      }
21
22      //添加数据到队列
23      public void addQueue(int num) {
24          if (isFull()) {
25              System.out.println("队列已满");
26              return;
27          }
28          arr[rear] = num;
29          rear = (rear + 1) % maxSize;
30      }
31
32      //获取队列的数据，出队列
33      public int getQueue() {
34          if (isEmpty()) {
35              System.out.println("队列为空，请先输入队列");
36              return 0;
37          }
38          int value = arr[front];
39          front = (front + 1) % maxSize;
40          return value;
41      }
42
43      //求出当前队列有效数据的个数
44      public int size() {
45          return (rear + maxSize - front) % maxSize;
46      }
47      //遍历队列
48      public void showQueue(){
49          if (isEmpty()) {
50              System.out.println("队列为空，请先输入队列");
51              return;
52          }
53          for (int i = front; i < front+size(); i++) {
54              System.out.printf("队列为：%d\t",arr[i%maxSize]);
55          }
56      }
57
```

```
58  }
```

JavaSE

子类与父类的执行顺序

```
1   class SuperClass {
2       public static void main(String[] args) {
3           new SubClass();
4       }
5       public SuperClass(){
6           System.out.println("Super");
7       }
8   }
9
10
11  class SubClass extends SuperClass{
12      public SubClass(){
13          this(1);
14          System.out.println("Sub");
15      }
16      public SubClass(int i){
17          System.out.println(i);
18      }
19  }
```

```
1   Super
2   1
3   Sub
```

```
1   public class Test {
2       public static void main(String[] args) {
3           SuperClass s1 = new SubClass();
4       }
5   }
6   class SuperClass{
7       public int i = 1;
8       public SuperClass(){
9           System.out.println("SuperClass");
10          System.out.println(getI());
11      }
12
13      public int getI() {
14          return i;
15      }
```

```
}
class SubClass extends SuperClass{
    public int i = 2;
    public SubClass(){
        System.out.println("SubClass");
        System.out.println(getI());
    }
}
```

```
SuperClass
1
SubClass
1
```

```
public class Test {
    public static void main(String[] args) {
        SuperClass s1 = new SubClass();
    }
}

class SuperClass{
    public int i = 1;
    public SuperClass(){
        System.out.println("SuperClass");
        System.out.println(getI());
    }

    public int getI() {
        return i;
    }
}


class SubClass extends SuperClass{
    public int i = 2;
    public SubClass(){
        System.out.println("SubClass");
        System.out.println(getI());
    }

    @Override
    public int getI() {
        return i;
    }
}
```

```
SuperClass
0
SubClass
2
```

```
public class Test {
    public static void main(String[] args) {
        SuperClass s1 = new SubClass();
        s1.print();
    }
}
class SuperClass{
    public static void print(){
        System.out.println("Super");
    }
}

class SubClass extends SuperClass{
    public static void print(){
        System.out.println("Sub");
    }
}
```

```
Super
```

```
class HelloA {
    public HelloA() {
        System.out.println("HelloA");
    }
    {
        System.out.println("I'm A class");
    }
    static {
        System.out.println("static A");
    }
}

class HelloB extends HelloA {
    public HelloB() {
        System.out.println("HelloB");
    }
    {
        System.out.println("I'm B class");
    }
    static {
```

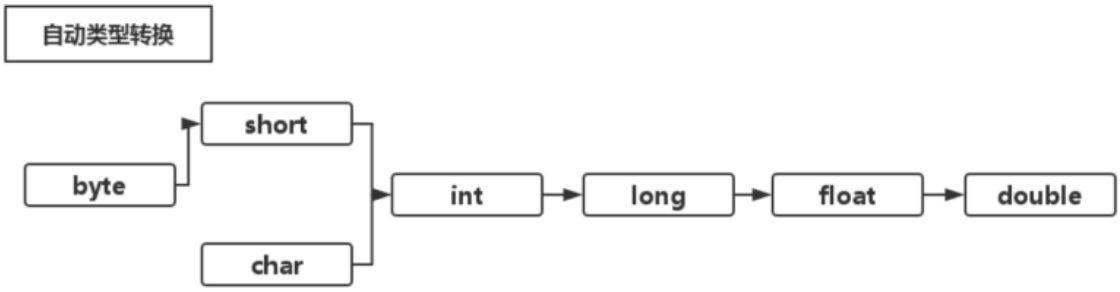
```
        System.out.println("static B");
    }
    public static void main(String[] args) {
        new HelloB();
    }
}
```

```
static A
static B
I'm A class
HelloA
I'm B class
HelloB
```

执行子类的构造方法时，JVM先执行一次父类的构造方法。

- 普通代码块----从属于对象，只有创建对象时才会开辟空间执行代码。
- 静态代码块----优先创建对象时，在方法区内开辟空间。
- 三者的优先级顺序： 静态代码块 > 普通代码块 > 构造方法

自动类型提升



以下代码的执行结果是（ B ）。（选择一项） ???

```
boolean m = false;
if(m = false){
System.out.println("false");
}else{
System.out.println("true");
}
```

A.	false
B.	true
C.	编译错误
D.	无结果

1、	**以下关于this和super关键字的说法错误的是（ BD ）。（选择二项）**	
	A.	this关键字指向当前对象自身，super关键字指向当前对象的直接父类
	B.	在main方法中可以存在this或super关键字，但不能同时存在。
	C.	this和super关键字都可以访问成员属性，成员方法和构造方法
	D.	在一个类的构造方法中可以同时使用this和super来调用其他构造方法 解析：此题考点----this super关键字 选项A：this代表本类对象的一个引用，super代表本类直接父类的一个引用 选项B：静态方法、静态代码块等，不允许使用this super关键字。 因为：静态方法等优先开辟空间，在使用this时，对象可能还没有 创建，所以JVM不允许在静态方法、静态代码块中使用他们。 选项C：this super代表的是一个引用，可以调用类的属性和方法。 选出D：类的构造方法，首行要么是使用this，要么使用super，只能是二选一

3、	**下列选项中关于Java中super关键字的说法正确的是（　　AD　）。（选择二项）**	
	A	super关键字是在子类对象内部指代其父类对象的引用
	B.	super关键字不仅可以指代子类的直接父类，还可以指代父类的父类
	C.	子类通过super关键字只能调用父类的方法，而不能调用父类的属性
	D.	子类通过super关键字可以调用父类的构造方法 解析：此题考点-----super关键字 选项A：　super作为子类直接父类对象的一个引用存在，正确； 选项B：super只能引用子类的直接父类对象，不能一次引用父类的父类。 选项C：为了区分父类和子类重复的属性或方法时，可以使用super进行分区 选项D：允许使用super调用父类的构造方法，则必须是在首行进行调用。

5、父类Person中定义了一个private void show()的方法，那么子类要重写这个方法时，方法的访问修饰符可以是默认的，protected或public。（　　×　）

解析：父类私有的方法，子类不能继承，同时也不能重写。

方法的重写，是建立在继承的基础之上，既然不能继承就无谈重写。

但是私有方法，允许在本类中进行重载，对权限修饰符没有限制。

此题考点----final关键字

可以修饰类，此类变为终结类，不能有子类。例如：System Math

可以修饰变量，此变量变为常量，值不能再改变。

可以修饰方法，此方法不能被重写，允许被重载。

可以修饰对象，此对象引用地址不能再改变，但是对象的值允许改变。

3.	**以下代码中错误的语句是（ D ）。（选择一项）**	

	public class Something{undefined
	public static void main(String[] args){undefined
	final Other o=**new** Other();
	new Something().addOne(o);//1
	}
	public void addOne(Other o){undefined
	o.i++;//2
	o = **new** Other();//3
	}
	}
	class Other{undefined
	public int i;
	}	

	A	1
	B.	2
	C.	3
	D.	没有错误

2、	编译并运行如下Java程序，将输出（　　D ）。（选择一项）	
	public static void main(String[] args) {undefined **try** {undefined **int** num1 = 2; **int** num2 = 0; **int** result = num1 / num2; System.*out*.println(result); **throw new** NumberFormatException( ); } **catch** (ArrayIndexOutOfBoundsException e) {undefined System.*out*.print("1"); } **catch** (NumberFormatException e) {undefined System.*out*.print("2"); } **catch** (Exception e) {undefined System.*out*.print("3"); } **finally** {undefined System.*out*.print("4"); } System.*out*.print("5"); }	
	A	134
	B.	2345
	C.	1345
	D.	345 解析：此题考点-----捕捉异常 从以上代码发现，**int** result = num1 / num2; 被除数为零，发生异常 ArithmeticException算术异常，后续的输出语句不再执行，进入到catch 代码块寻找匹配的异常类型，发现没有catch代码块中没有ArithmeticException算术异常，只要让父类异常类型代替子类接收，此处体现了对象的多态-----对象的向上转型；执行此catch块中的代码，打印输出3， 再进行分析，catch块进行匹配了之后，执行finally代码块，输出4， 还有最后一句System.*out*.print("5");它不再try catch finally中，最后执行它， 输出5，所以------最后结果：345