Object-Oriented Programming with Java

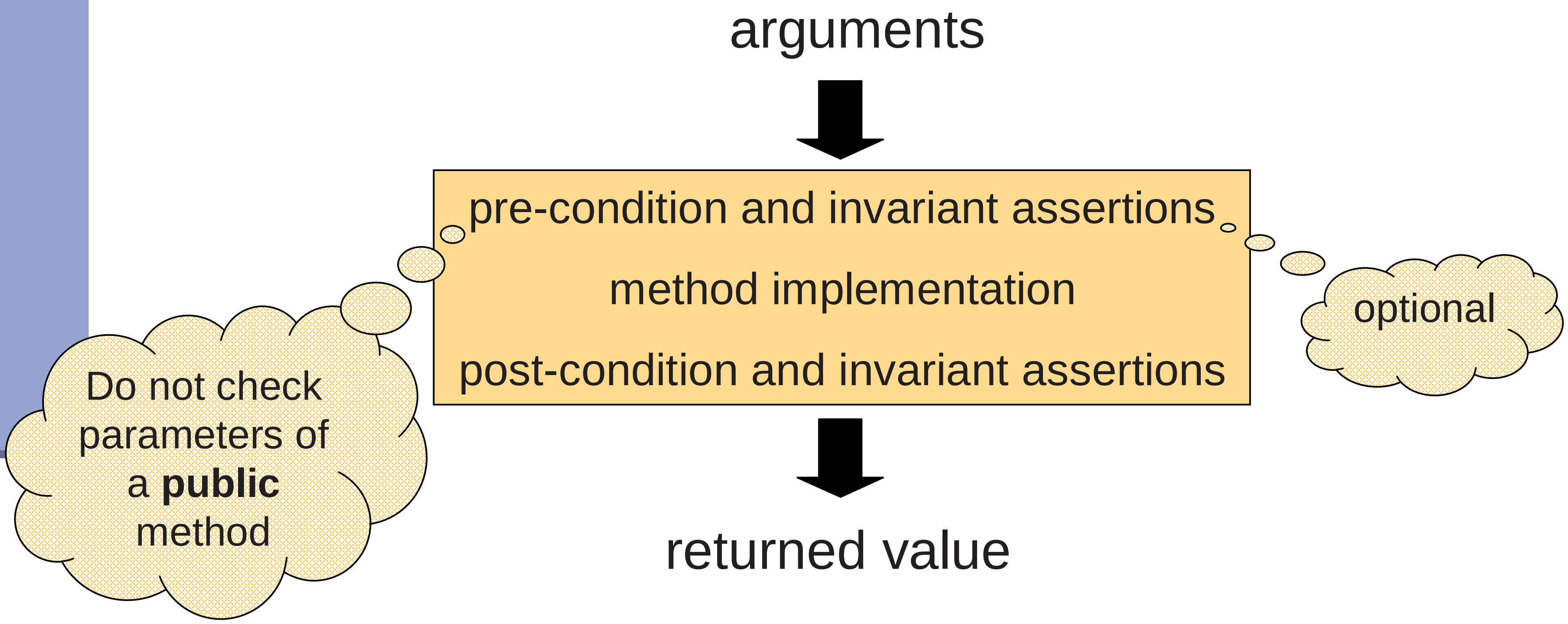
Recitation No. 6 (Assertions, Proxies and more)

Assertions

- Boolean expressions
- State properties that must be satisfied at certain stages of program execution
- Example: `i ≥ 0 && i < CAPACITY;`
- Useful for:
  - contract specification and documentation (specifying invariant & pre/post conditions)
  - unit testing (an aspect of defensive programming)

Using Assertions

- Checking method correctness



arguments
pre-condition and invariant assertions
method implementation
post-condition and invariant assertions
returned value
Do not check parameters of a **public** method
optional

Using Assertions (cont.)

- Checking internal invariants:
  - of `if-else` statements

```
if (num % 3 == 0) {
    ...
} else if (num % 3 == 1) {
    ...
} else {
    assert(num % 3 ==2)
}
```

```
public void assert(boolean e)
    throws AssertionError;
```

Using Assertions (cont.)

- of switch statements with no default case

```
switch(traffic_light) {
    case Color.RED:
          ...
          break;
    case Color.GREEN:
          ...
          break;
}   default:
          assert(false);
}
```

- of loop statements

Assertion Errors vs. Exceptions

- Both catch problems in the program
- The intended usage is different:
  - Assertion errors indicate code bugs
  - Exceptions are for the user
- Use Exceptions if something might go wrong and you have no control over **within the class**
  - e.g. IO, arguments of a public method

Assertion Mechanism Design

- Assertions are usually turned off in released versions:
  - They may be time consuming
  - Rarely allow error recovery
  - No helpful error message to the user
- Suggest a mechanism that allows one to enable or disable pre/post condition and class invariant assertions at runtime

Solution 1: Using a Flag

```
public void add(String s) {
     if (assertFlag)
          assert(...)

     implementation

     if (assertFlag)
          assert(...)
}
```

- But, what if there are many exit points?

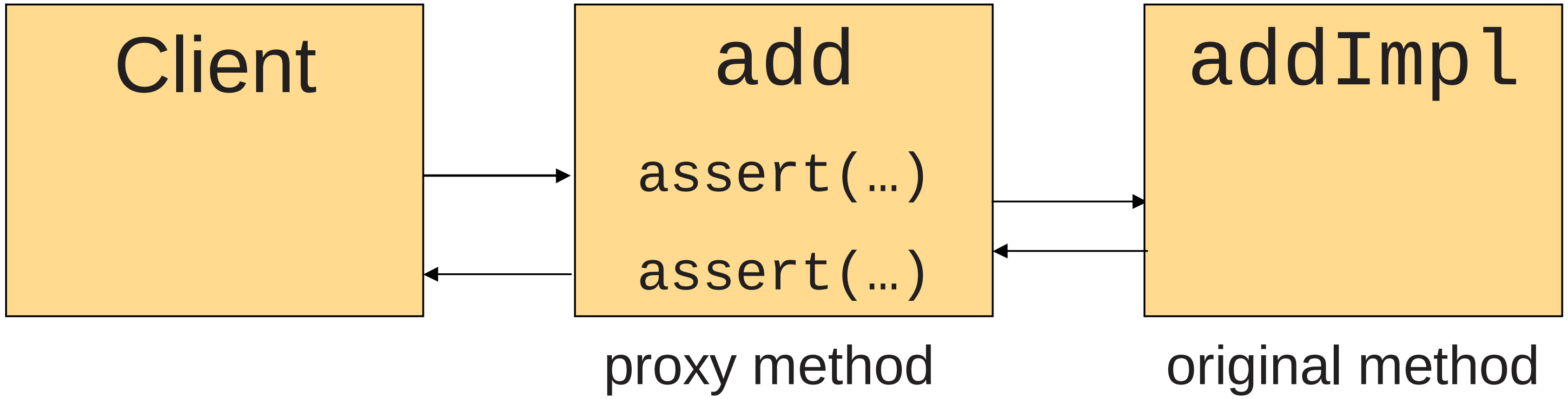
```
if () {
    ...
    return;
} else if {
    ...
    return;
}
...
```

Invariant and post-condition should be checked at each exit point

Solution 2: Proxy Methods

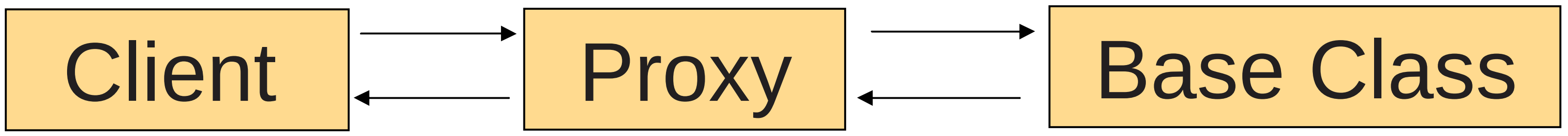
```
public void add(String s) {
        assert(...)
        addImpl(s);
        assert(...)
}
```

Internal method



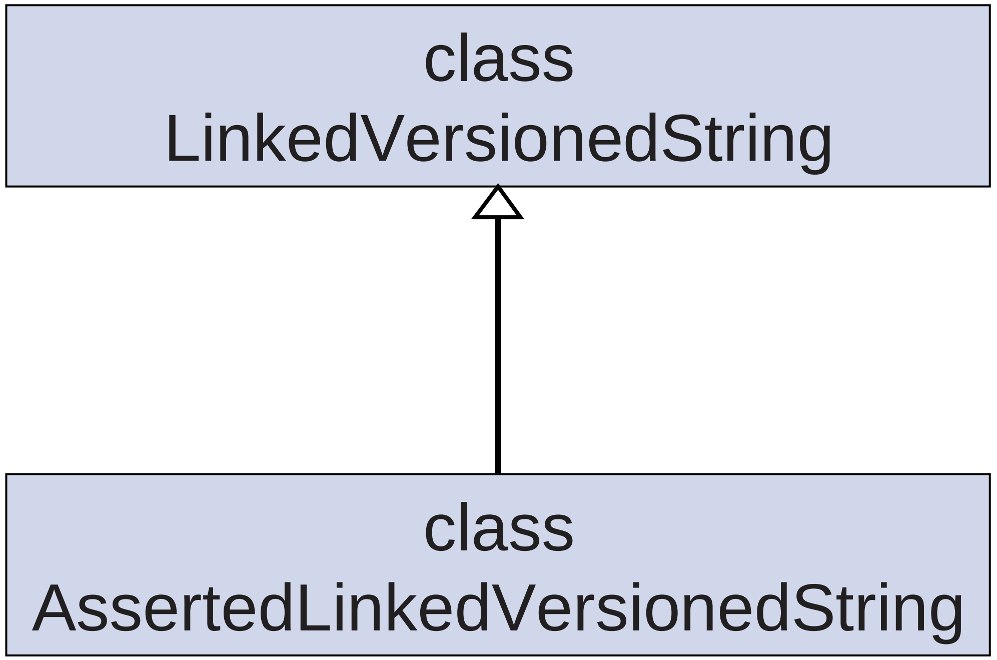
Solution 3: A Proxy Class

- Extends the proxy method idea, but asserted and regular methods are in different classes.
- The Proxy class:
  - acts as a surrogate for the base class
  - has the same interface as the base class
- A factory can be used to choose between proxy and base classes.


$$\begin{tikzcd}\text{Client} \arrow[r, shift left] & \text{Proxy} \arrow[l, shift left] \arrow[r, shift left] & \text{Base Class} \arrow[l, shift left]\end{tikzcd}$$

Inherited Proxy Class

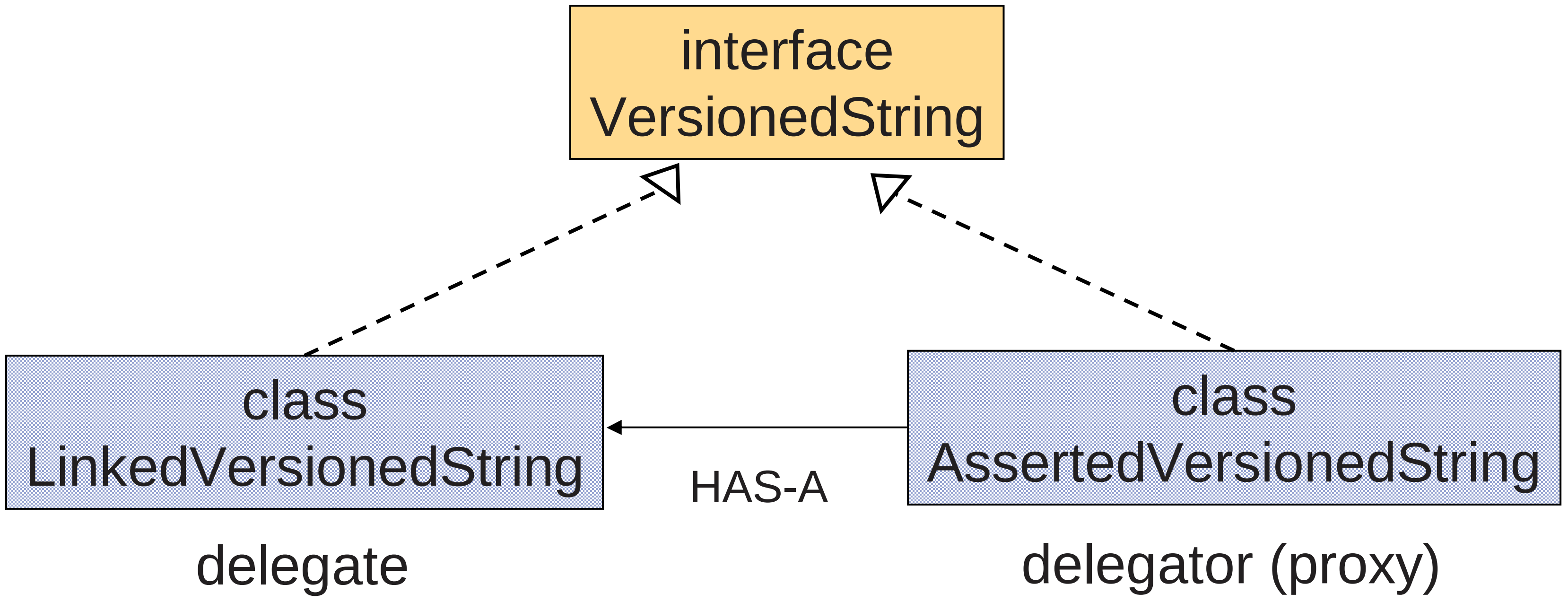
- IS-A relationship



```
public class AssertedLinkedVersionedString
    extends LinkedVersionedString {
    public void add (String s) {
        assert(…)
        super.add(s);
        assert(…)
    }
    ...
}
```

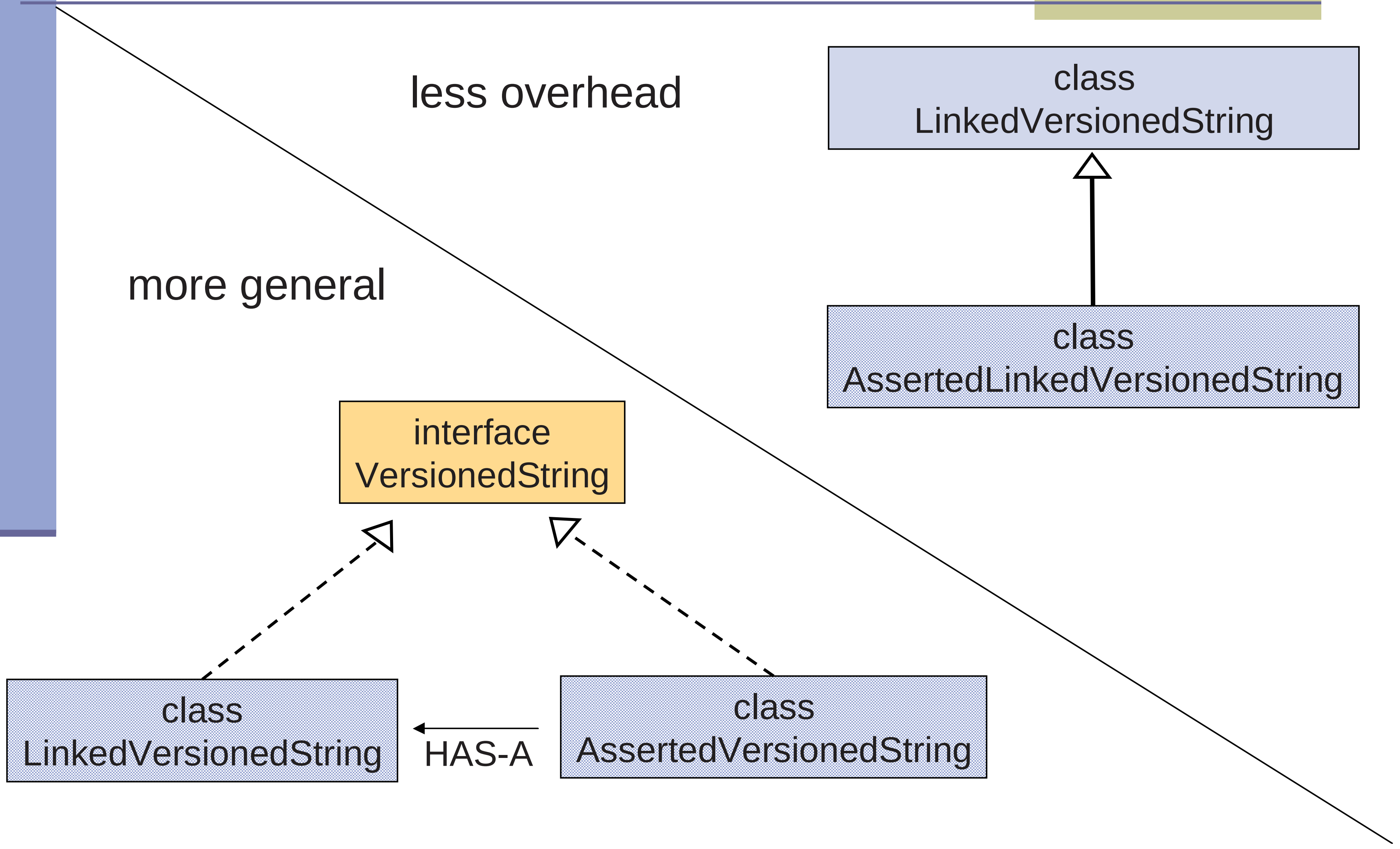
Delegating Proxy Class

- HAS-A relationship instead of IS-A



```
public class AssertedLinkedVersionedString
    implements VersionedString {
    **VersionedString vstring;**
    public void add (String s) {
            assert(…)
            **vstring.add(s);**
            assert(…)
    }
    ...
}
```

Inheritance vs. Delegation



Java Assertion Facility

- From Java1.4 on
- Assertion Statements:
  - `assert booleanExpression;`
  - `assert booleanExpression : messageExpression;`
- If `booleanExpression` is `false`, an `AssertionError` is thrown.
- Use `assert` only in an executable code

Java Assertion Facility (cont.)

- `booleanExpression` can be a call to a method:

  ```
  assert postCondition() && invariant();
  ```

- Assertions are disabled by default
- Assertions are enabled at runtime by `enableassertions` or `-ea`

Another Proxy Example

- Consider an Internet Service Provider whose clients often access the same web pages, resulting in multiple copies of web files transmitted via its server.
- How can we improve this situation?
- Use a Cache Proxy!

Other Proxy Examples

- Access Proxy
- Firewall Proxy
- Virtual Proxy (Lazy Proxy)
- Remote Proxy
- Synchronization Proxy
- Smart Reference Proxy

A Word about Interfaces

- An interface can extend several interfaces
- Interface methods are by definition public and abstract:

```
public interface MyInterface {
    public abstract int foo1(int i);
    int foo2(int i);
}
```

The type of foo1 and foo2 is the same.

Visibility

	class	subclasses	package	other
private	X	-	-	-
package	X	-	X	-
protected	X	X*	X	-
public	X	X	X	X

Default

Visibility (cont.)

```
package A;

public class Molecule {
    …
    protected void calculateWeight() {
        …
    }
    …
}
```

```
package B;

public class Protein extends
    Molecule {
    void foo(Protein p, Molecule m)
    {
        calculateWeight();
        p.calculateWeight();
        m.calculateWeight();
        …
    }
    …
}
```

Illegal

Initialization

```
public class Test {
    private int a = getB();
    private int b = 5;

    private int getB() {
        return b;
    }

    public static void main(String args[]) {
        System.out.println((new Test()).a);
    }
}
```

The output is:
0

compile? If no, why?
hrow a runtime exception?
If yes, why? If no, what is the output?

Initialization

```
public class Test {
    private int b = 5;
    private int a = getB();

    private int getB() {
        return b;
    }

    public static void main(String args[]) {
        System.out.println((new Test()).a);
    }
}
```

The output is:
5

compile? If no, why?
hrow a runtime exception?
If yes, why? If no, what is the output?

Initialization

```
public class Foo {
    static int bar;

    public static void main (String args []) {
        bar += 1;
        System.out.println("bar = " + bar);
    }
}
```

The output is:
1

mpile? If no, why?
ow a runtime exception?
If yes, why? If no, what is the output?

Exceptions

```
int i=1, j=1;
try {
     i++;
      j--;
      if (i/j > 1)
             i++;
} catch(ArithmeticException e) {
     System.out.println(1);
} catch(Exception e) {
     System.out.println(2);
} finally {
     System.out.println(3);
}
```

The output is:

```
1
3
```