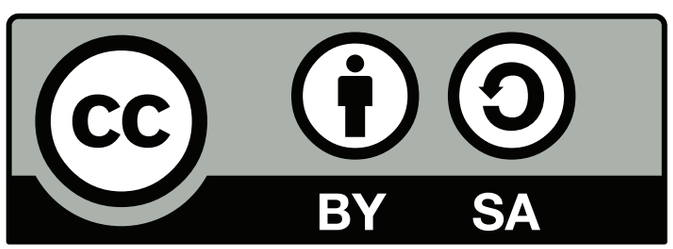
Tecnologia e Applicazioni Internet 2010/11

Lezione 1 - Gestione delle dipendenze

Matteo Vaccari
http://matteo.vaccari.name/
vaccari@pobox.com



Modalità di esame

- Preparazione di un elaborato, da solo o in coppia
- Discussione orale dell'elaborato

Che cosa rende il codice *difficile da testare?*

Mescolare *new* e *logica*

```
@Test
public void shouldRecognizeExpiredOption() {
    Date expiration = dateAt(1995, MAY, 28);

    Option option = new Option(expiration);

    assertTrue(option.isExpired());
}
```

```
public class Option {

    private final Date expiration;

    public Option(Date expiration) {
        this.expiration = expiration;
    }

    public boolean isExpired() {
        Date now = new Date();
        return expiration.before(now);
    }
}
```

Roberto Albertini, Sourcesense

```java
@Test
public void shouldRecognizeExpiredOption() {
    Date expiration = dateAt(2008, MAY, 28);

    Option option = new Option(expiration);

    assertTrue(option.isExpired());
}
```

```java
public class Option {

    private final Date expiration;

    public Option(Date expiration) {
        this.expiration = expiration;
    }

    public boolean isExpired() {
        Date now = new Date();
        return expiration.before(now);
    }
}
```

```java
@Test
public void shouldRecognizeNonExpiredOption() {
    Date expiration = dateAt(2020, MAY, 28);

    Option option = new Option(expiration);

    assertFalse(option.isExpired());
}
```

Per quanto tempo funzionerà?

Come posso testare i casi limite?

Roberto Albertini, Sourcesense

`new Date()` non è programmabile

```
public boolean isExpired() {
  Date now = new Date();
  return expiration.before(now);
}
```

lo sostituisco con un collaboratore

```
public boolean isExpired() {
  Date now = clock.now();
  return expiration.before(now);
}
```

Roberto Albertini, Sourcesense

```
private final Date expiration;

public Option(Date expiration) {
   this.expiration = expiration;
}
```

devo esplicitare la dipendenza
rendere iniettabile il collaboratore

```
private final Date expiration;
private final Clock clock;

public Option(Date expiration, Clock clock) {
   this.expiration = expiration;
   this.clock = clock;
}
```

Roberto Albertini, Sourcesense

Il codice di produzione

```
public class Option {

    private final Date expiration;
    private final Clock clock;

    public Option(Date expiration, Clock clock) {
        this.expiration = expiration;
        this.clock = clock;
    }

    public Option(Date expiration) {
        this(expiration, new RealClock());
    }
}
```

```
public interface Clock {
    Date now();
}
```

```
public class RealClock implements Clock {

    public Date now() {
        return new Date();
    }

}
```

Roberto Albertini, Sourcesense

Il codice di test

```
public class OptionTest {

    @Test
    public void shouldRecognizeExpiredTask() {
        Clock clockAt2009028 =
            new ProgrammableClock(
                dateAt(2009, MAY, 28));

        Date expiration = dateAt(2009, MAY, 27);

        Option option =
                new Option(expiration, clockAt20090528);

        assertTrue(option.isExpired());
    }
}
```

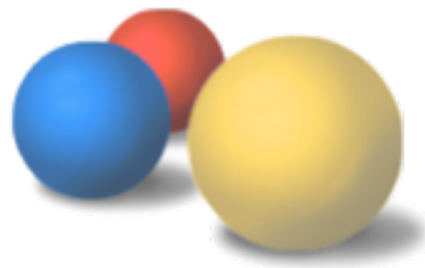
```
public interface Clock {
    Date now();
}
```

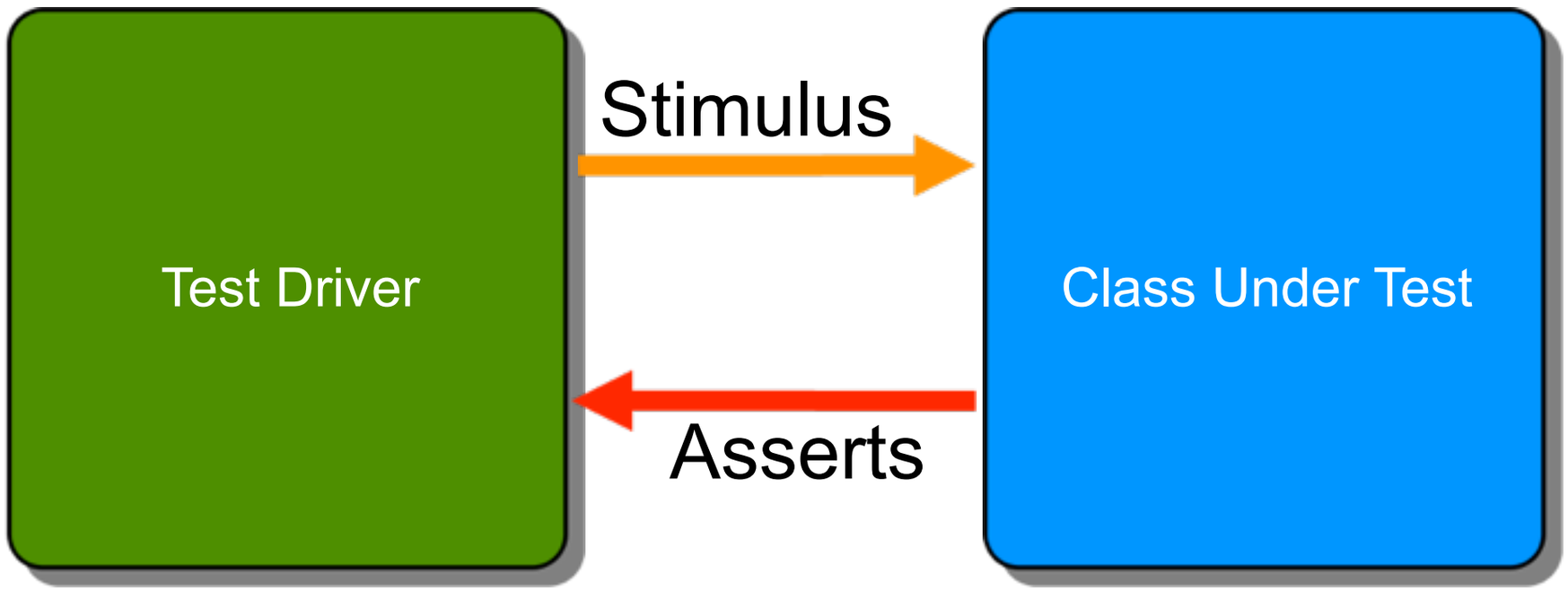
```
public class ProgrammableClock
                        implements Clock {
    private Date now;

    public ProgrammableClock(Date date) {
        this.now = date;
    }

    public Date now() {
        return now;
    }
}
```

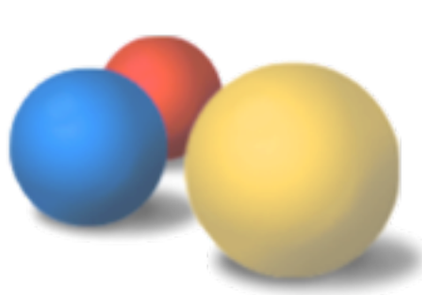
Unit Testing a Class

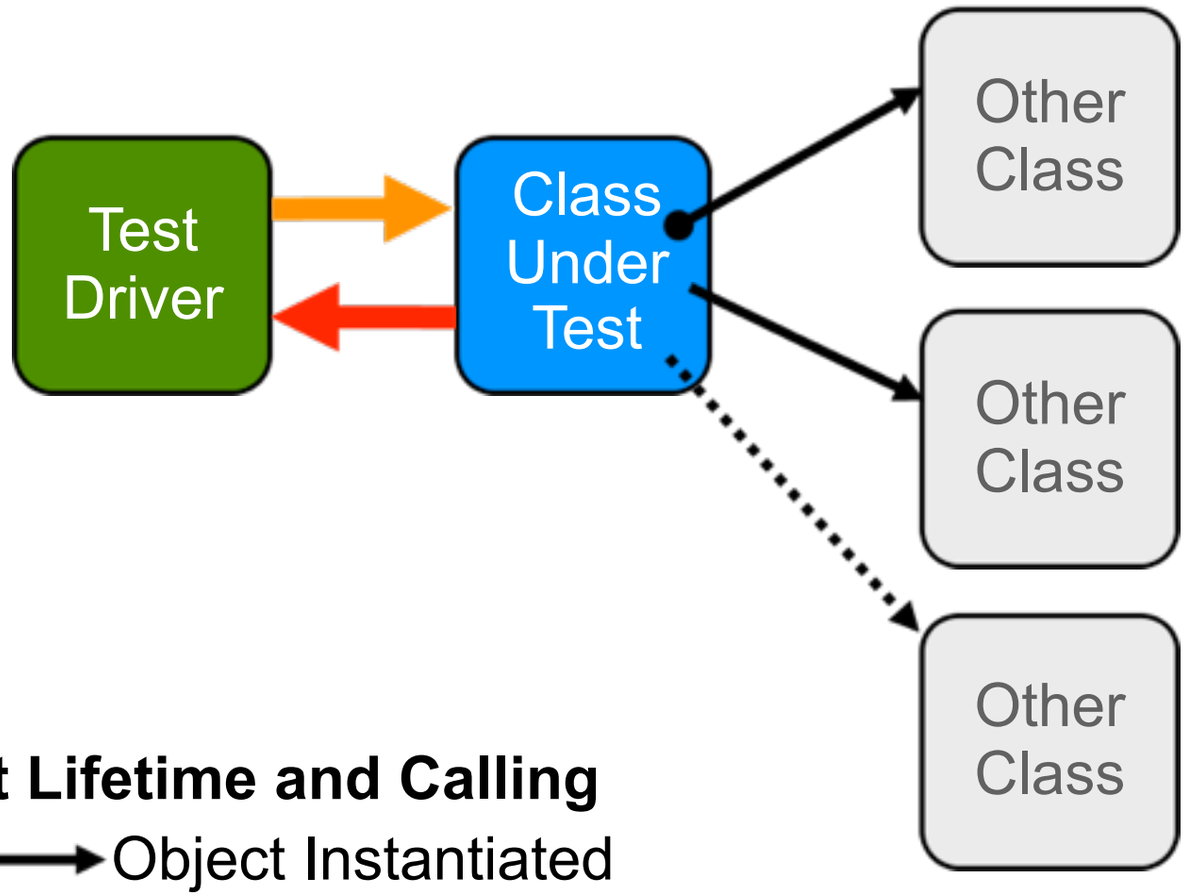




Miško Hevery Google

Unit Testing a Class



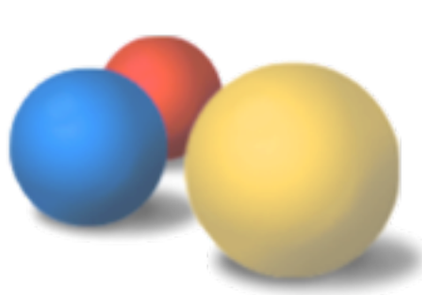


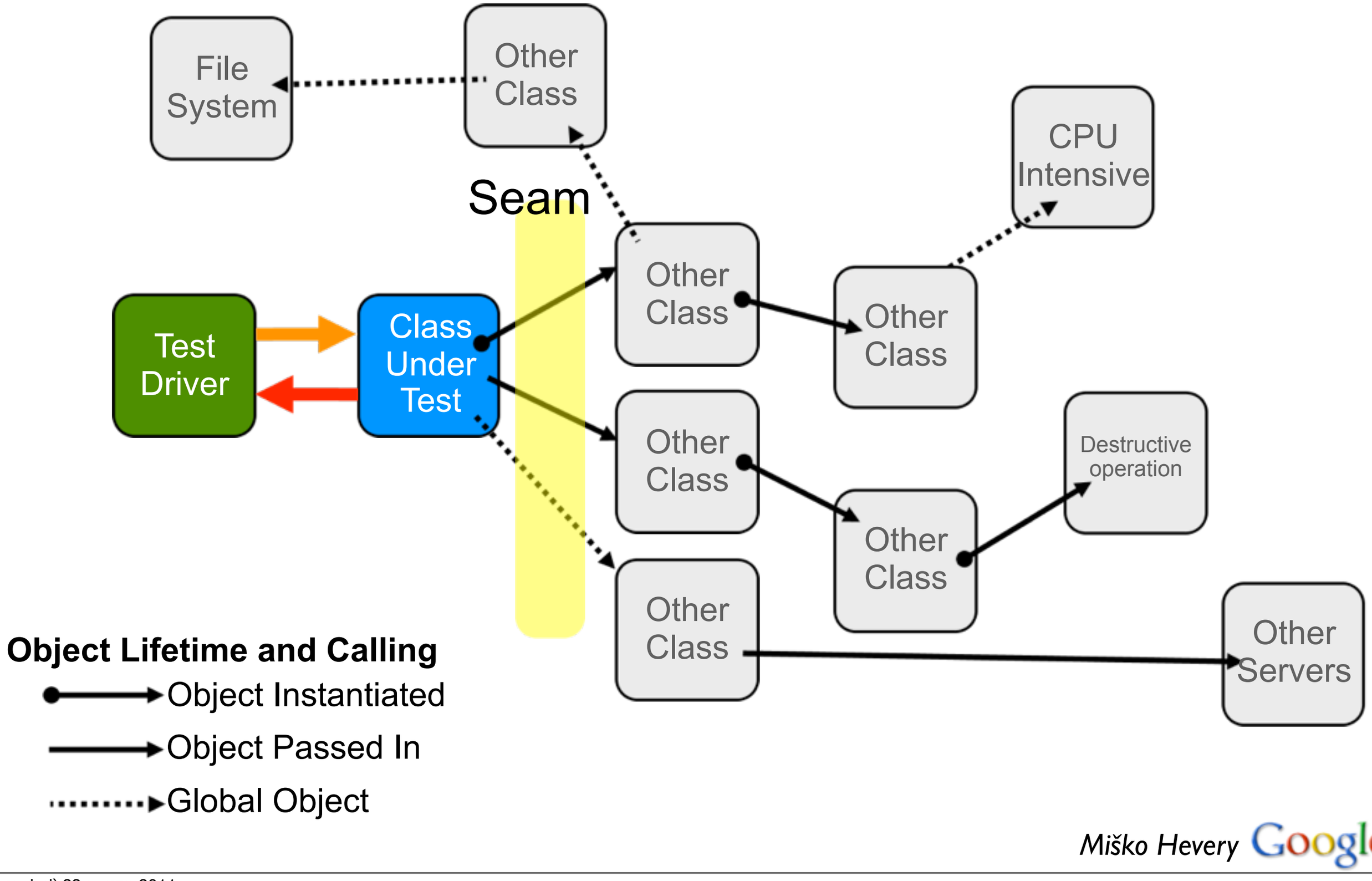
Object Lifetime and Calling

- Object Instantiated
- Object Passed In
- Global Object

Miško Hevery Google

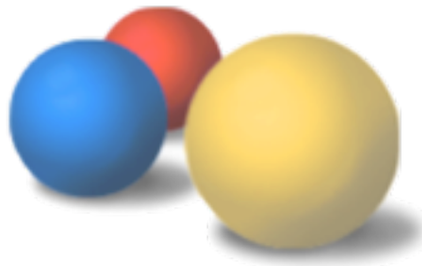
Unit Testing a Class

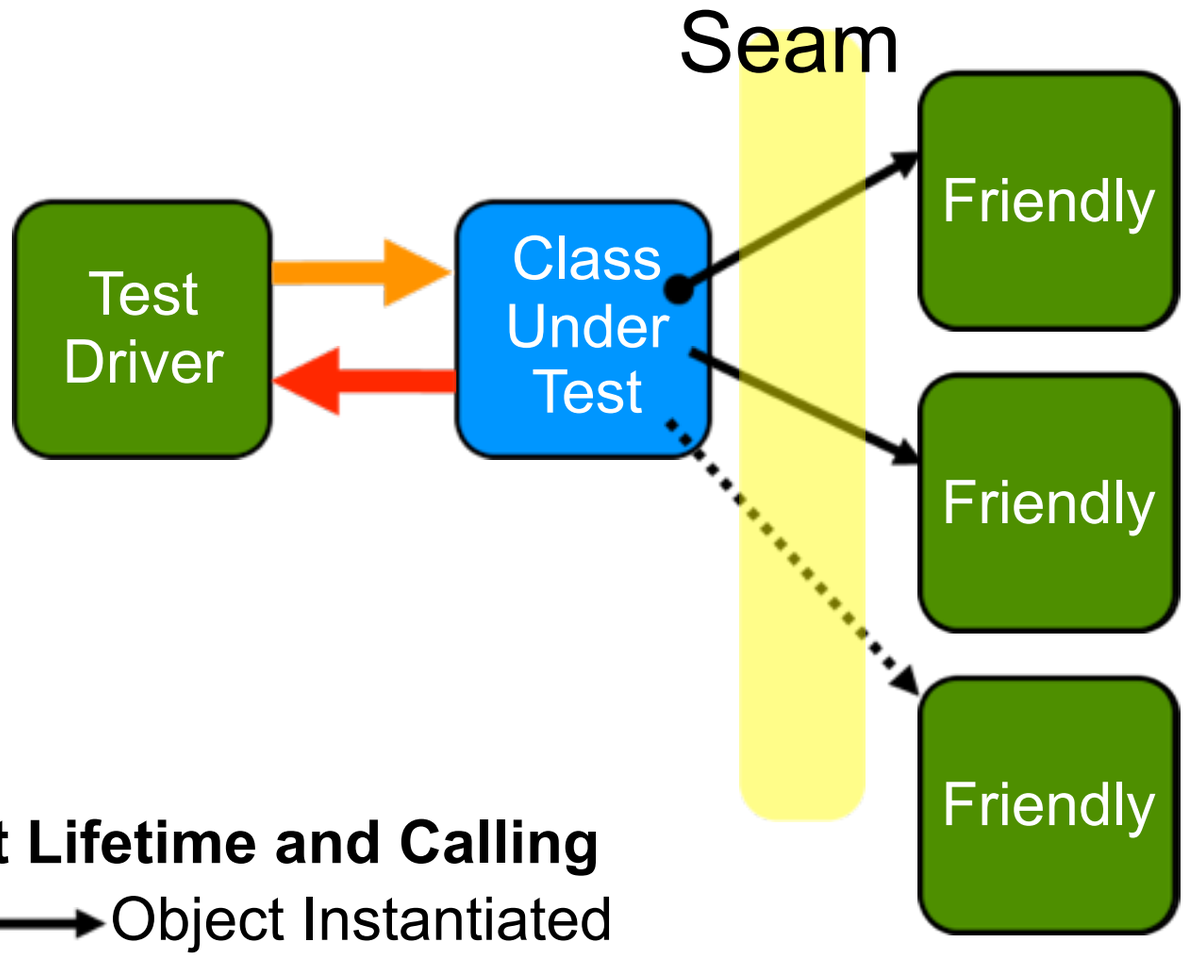




Miško Hevery Google

Unit Testing a Class

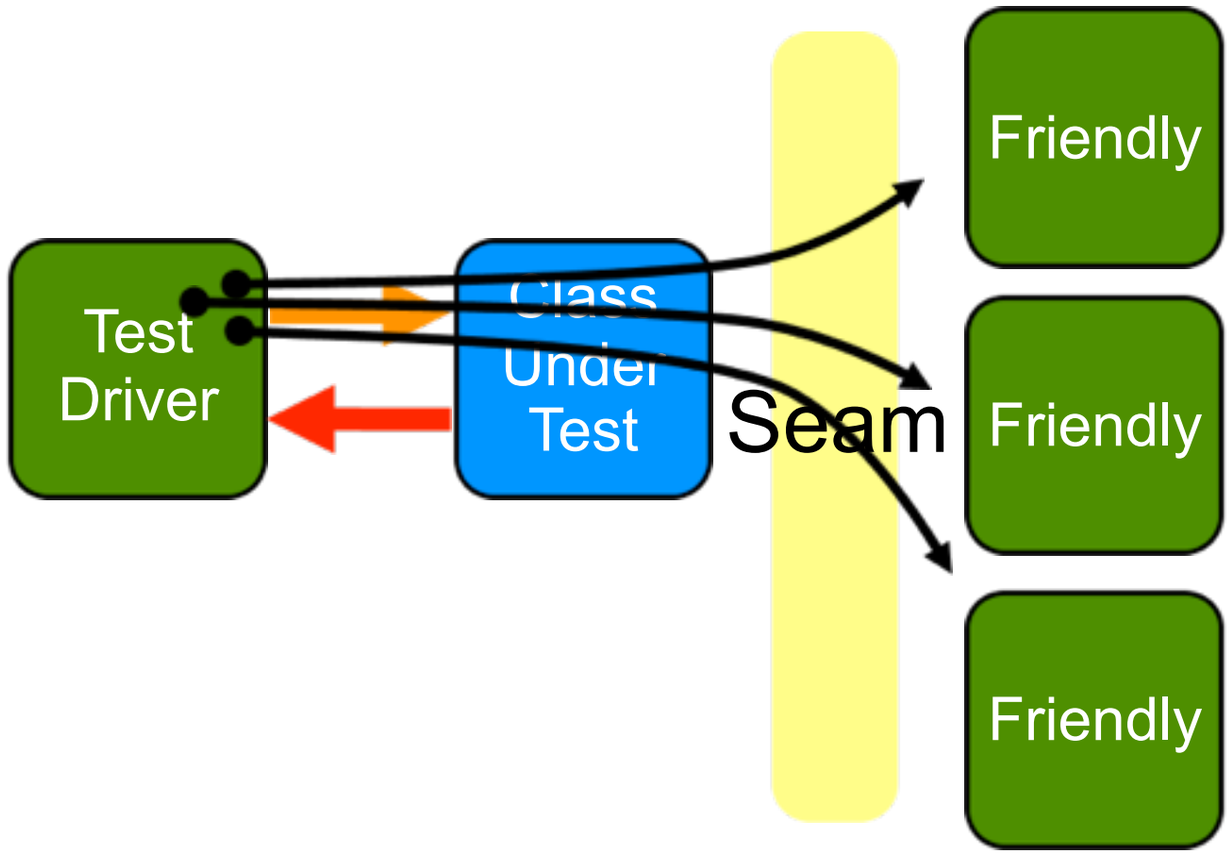




Object Lifetime and Calling

- Object Instantiated
- Object Passed In
- Global Object

Unit Testing a Class



Object Lifetime and Calling

- Object Instantiated
- Object Passed In
- Global Object

Miško Hevery Google

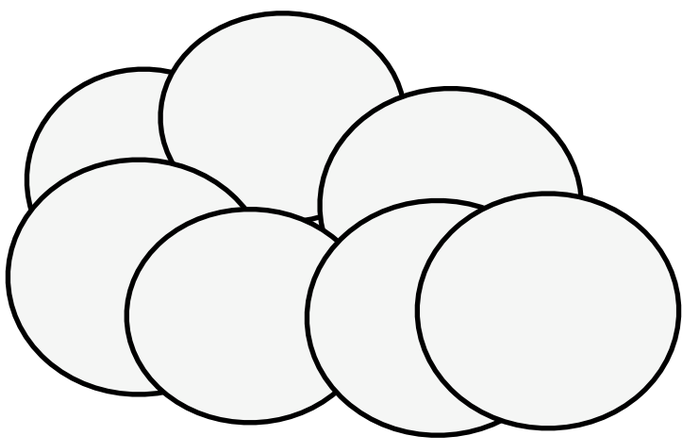
Two piles

Pile of Objects

- Business logic
- This is why you're writing code

Pile of New Keywords

- Provider<T> objects
- Factories
- Builders
- This is how you get the code you write to work together

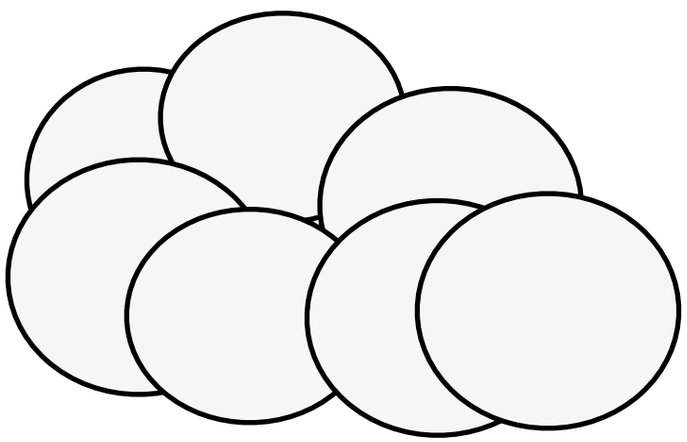


new new new new new new new new new new new new new

Two piles

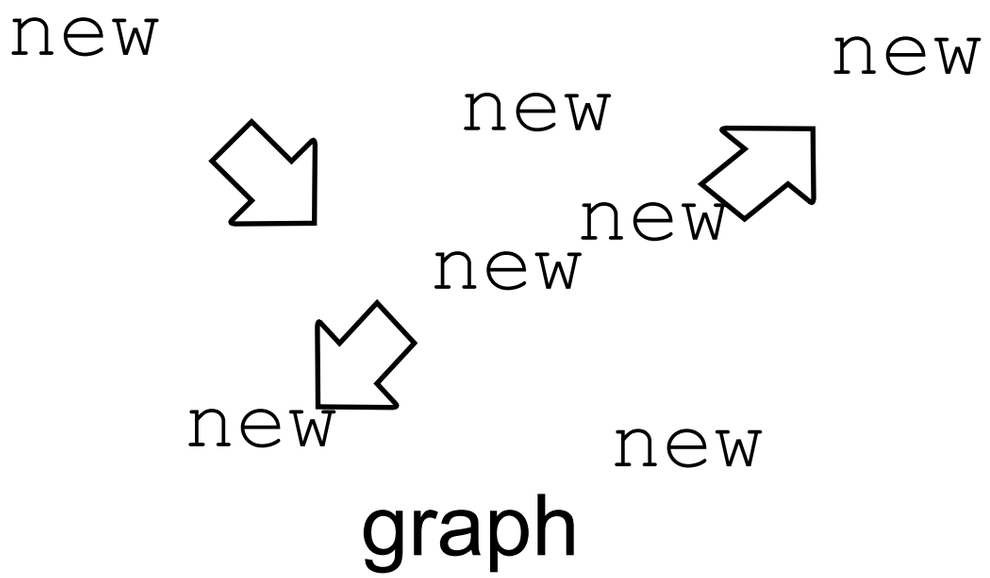
Pile of Objects

- Responsibility is business logic, domain abstractions



Pile of New Keywords

- Responsibility is to build object graphs



Miško Hevery Google

La struttura di un *main*

```
public static void main(String[] args) throws Exception {
    // Creation Phase
    Server server = new ServerFactory(args).createServer();

    // Run Phase
    server.start();
}
```

Stato 

- Singleton
- Metodi statici
- JNDI
- Service locators

Stato GLOBALE

Rende le esecuzioni inconsistenti

eseguo due volte e ottengo risultati diversi

Nasconde le collaborazioni

non so più quali da quali cose dipende la classe sotto test

Diventa rilevante l'ordine dei test

o molto complicate le setup e teardown dei test

Come si cura

Non si usano mai variabili globali

non sempre è possibile

ad esempio il db è una variabile globale

Come si cura

si introduce una dipendenza iniettabile

il nuovo oggetto sarà
responsabile dell'accesso allo stato globale

nei test si inietta una istanza "di test"

Cavalcare i collaboratori

Legge di Demetra

```
dog.getBody().getTail().wag();
```

Tell, Don't Ask!

```
dog.expressHappiness();
```

Legge di Demetra

"*talk only to your friends*"

Ogni metodo M di un oggetto O
può invocare solo i metodi dei seguenti tipi di oggetti:

1. dei propri campi
2. dei parametri passati al metodo
3. di ogni oggetto che crea

Violazione della Legge di Demetra

Rende difficile trovare un errore
evidenziato da una barra rossa

molti oggetti sono coinvolti nell'esecuzione

Complica il setup dei test

devo preparare tutte le combinazioni di stati
degli oggetti coinvolti

Violazione della Legge di Demetra

Rende la classe accoppiata a tutte le interfacce attraversate dalla catena di punti

posso arrivare ad una classe non testabile

Come si cura

seguendo il principio "*Tell, Don't Ask*"

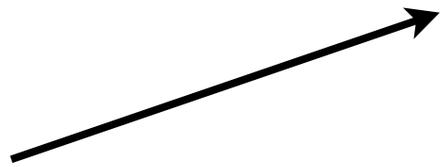
i “getter” vengono sostituiti

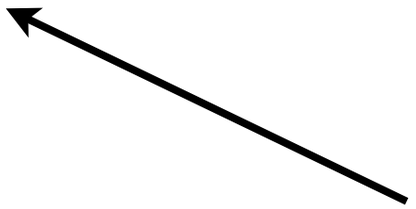
con con metodi che invocano delle azioni

piuttosto che chiedere altri oggetti

Introduction to Java I/O

Streams





byte streams

character streams

InputStream,
OutputStream

Reader, Writer

java.io.InputStream

```
public abstract class InputStream {
    /**
     * @return     the next byte of data, or <code>-1</code> if the end of the
     *             stream is reached.
     */
    public abstract int read() throws IOException;

    /**
     * Closes this input stream and releases any system resources associated
     * with the stream.
     */
    public void close() throws IOException {}

    // ...
}
```

java.io.Reader

```
public abstract class Reader {
    /**
     * Read characters into a portion of an array.  This method will block
     * until some input is available, an I/O error occurs, or the end of the
     * stream is reached.
     *
     * @param      cbuf  Destination buffer
     * @param      off   Offset at which to start storing characters
     * @param      len   Maximum number of characters to read
     *
     * @return     The number of characters read, or -1 if the end of the
     *             stream has been reached
     *
     * @exception  IOException  If an I/O error occurs
     */
    abstract public int read(char cbuf[], int off, int len) throws IOException;
```

java.io.Reader

```
public abstract class Reader {
    /**
     * Read a single character.  This method will block until a character is
     * available, an I/O error occurs, or the end of the stream is reached.
     *
     * @return     The character read, as an integer in the range 0 to 65535
     *             (<tt>0x00-0xffff</tt>), or -1 if the end of the stream has
     *             been reached
     */
    public int read() throws IOException {
        char cb[] = new char[1];
        if (read(cb, 0, 1) == -1)
            return -1;
        else
            return cb[0];
    }

    /**
     * Close the stream.  Once a stream has been closed, further read(),
     * ready(), mark(), or reset() invocations will throw an IOException.
     * Closing a previously-closed stream, however, has no effect.
     *
     * @exception  IOException  If an I/O error occurs
     */
    abstract public void close() throws IOException;
}
```

java.io.Writer

```
public abstract class Writer {
    /**
     * Write a single character.
     */
    public void write(int c) throws IOException {
        writeBuffer[0] = (char) c;
        write(writeBuffer, 0, 1);
    }

    /**
     * Write an array of characters.
     */
    public void write(char cbuf[]) throws IOException {
        write(cbuf, 0, cbuf.length);
    }

    /**
     * Write a portion of an array of characters.
     */
    abstract public void write(char cbuf[], int off, int len) throws IOException;

    /**
     * Write a string.
     */
    public void write(String str) throws IOException {
        write(str, 0, str.length());
    }
```

java.io.Writer

```
/**
 * Flush the stream.
 */
abstract public void flush() throws IOException;

/**
 * Close the stream, flushing it first.  Once a stream has been closed,
 * further write() or flush() invocations will cause an IOException to be
 * thrown.  Closing a previously-closed stream, however, has no effect.
 */
abstract public void close() throws IOException;
}
```

Example: copying a character stream

```
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class CopyCharacters {
    public static void main(String[] args) throws IOException {
        Reader inputStream = null;
        Writer outputStream = null;

        try {
            inputStream = new FileReader("xanadu.txt");
            outputStream = new FileWriter("characteroutput.txt");

            int c;
            while ((c = inputStream.read()) != -1) {
                outputStream.write(c);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Hard to test!

Hard to test!

Line I/O

```
public class CopyLines {
    public static void main(String[] args) throws IOException {
        BufferedReader inputStream = null;
        PrintWriter outputStream = null;

        try {
            inputStream = new BufferedReader(new FileReader("xanadu.txt"));
            outputStream = new PrintWriter(new FileWriter("characteroutput.txt"));

            String l;
            while ((l = inputStream.readLine()) != null) {
                outputStream.println(l);
            }
        } finally {
            if (inputStream != null) {
                inputStream.close();
            }
            if (outputStream != null) {
                outputStream.close();
            }
        }
    }
}
```

Bridging byte streams to char streams

```
// reading chars from an input stream of bytes
public void fromBytesToChars(InputStream byteStream)
                throws UnsupportedEncodingException {
   InputStreamReader charStream = new InputStreamReader(byteStream, "UTF-8");
   // read from charStream
}

// writing chars to an output stream of bytes
public void fromCharsToBytes(OutputStream byteStream)
                throws UnsupportedEncodingException {
   OutputStreamWriter charStream = new OutputStreamWriter(byteStream, "UTF-8");
   // write to charStream
}
```

Standard charsets

Every implementation of the Java platform is required to support the following standard charsets.

US-ASCII

Seven-bit ASCII, a.k.a. ISO646-US, a.k.a. the Basic Latin block of the Unicode character set

`ISO-8859-1`
ISO Latin Alphabet No. 1, a.k.a. `ISO-LATIN-1`

`UTF-8`
Eight-bit UCS Transformation Format

`UTF-16BE`
Sixteen-bit UCS Transformation Format, big-endian byte order

`UTF-16LE`
Sixteen-bit UCS Transformation Format, little-endian byte order

`UTF-16`
Sixteen-bit UCS Transformation Format, byte order identifiableleied by an optional byte-order mark

http://download.oracle.com/javase/6/docs/api/java/nio/charset/Charset.html

Esercizio: clone di paste(1)

Problema: incollare file riga per riga

prova uno due tre	+	alfa beta gamma delta	→	provaalfa unobeta duegamma tredelta

Soluzione “veloce”

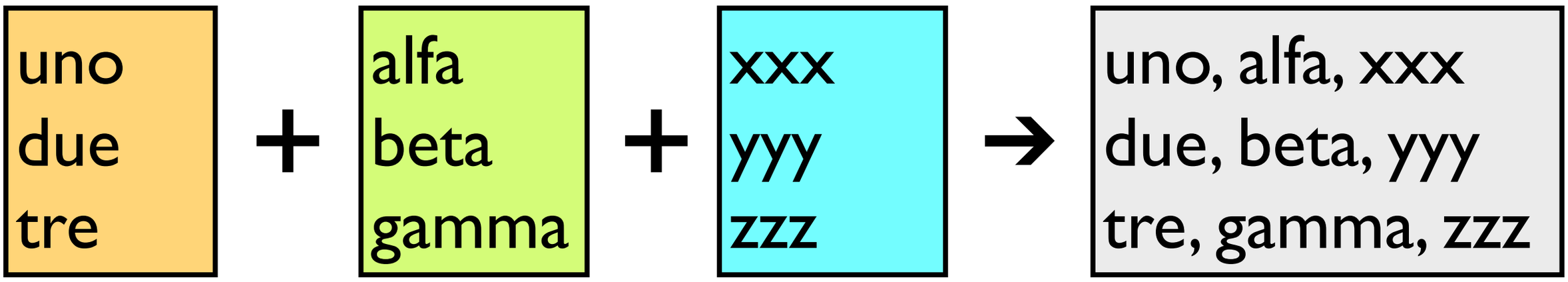
```
import java.io.*;

public class Main {

    public static void main(String ... args) throws IOException {
        BufferedReader first = new BufferedReader(new FileReader(args[0]));
        BufferedReader second = new BufferedReader(new FileReader(args[1]));
        BufferedWriter out = new BufferedWriter(new FileWriter(args[2]));
        String stringFromFirst, stringFromSecond;
        while ((stringFromFirst = first.readLine()) != null
                && (stringFromSecond = second.readLine()) != null) {
            out.append(stringFromFirst);
            out.append(stringFromSecond);
            out.newLine();
        }
        first.close();
        second.close();
        out.close();
    }
}
```


Quando abbiamo affrontato il problema anni fa, lo abbiamo risolto con una soluzione “veloce”

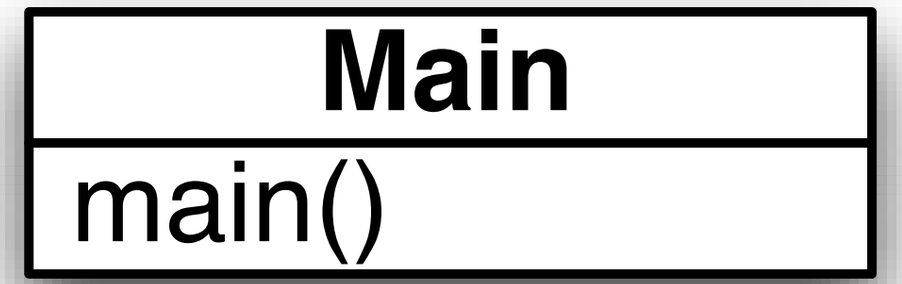
Nuovi requisiti!



- Numero arbitrario di file in ingresso
- Output (opzionalmente) separato da virgole
- Righe (opzionalmente) numerate

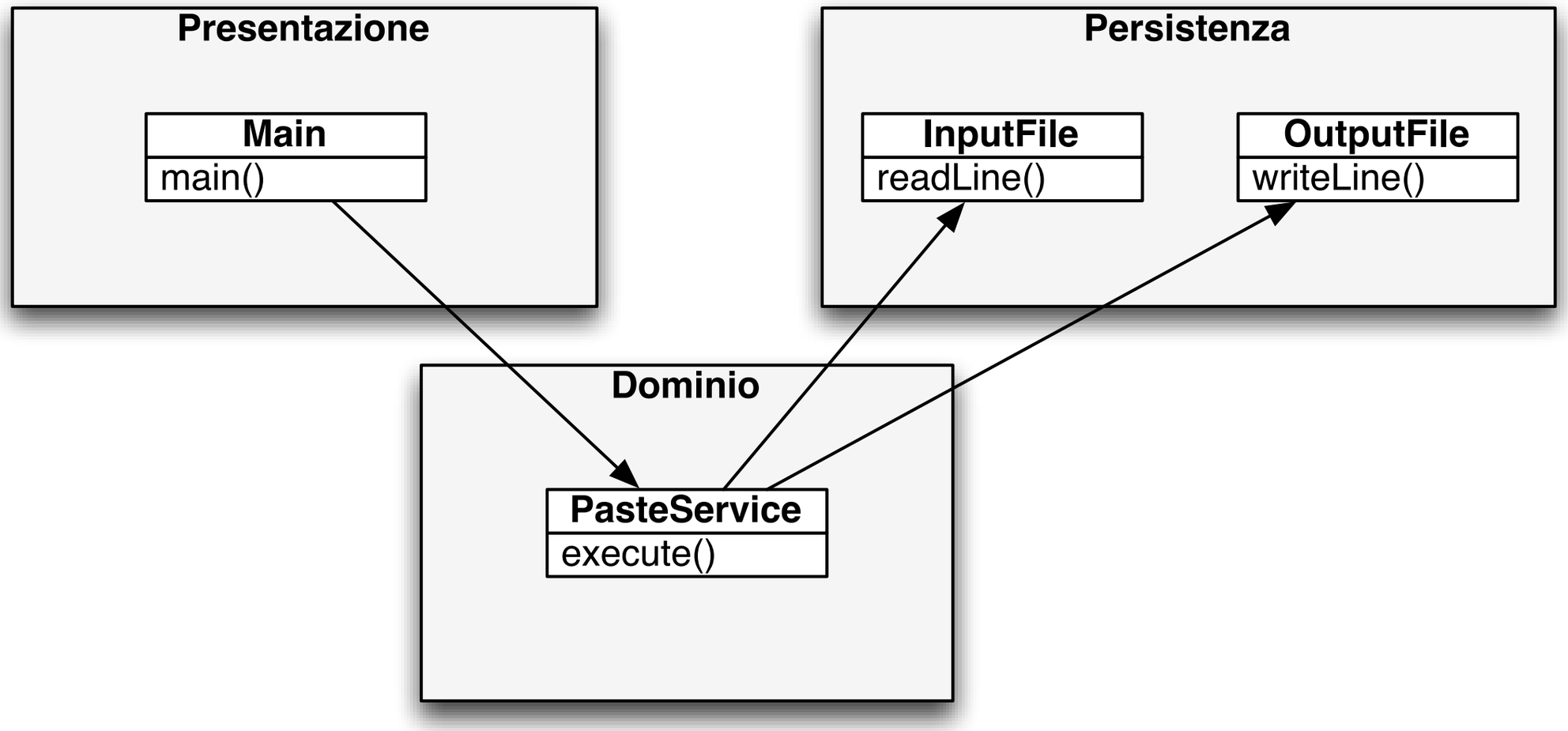
La soluzione “veloce” può avere risolto il problema originariamente. Ma quando dobbiamo, tempo dopo, risolvere un problema simile ma più complesso, adattare il vecchio software può risultare difficile.

Modello UML della versione “veloce”



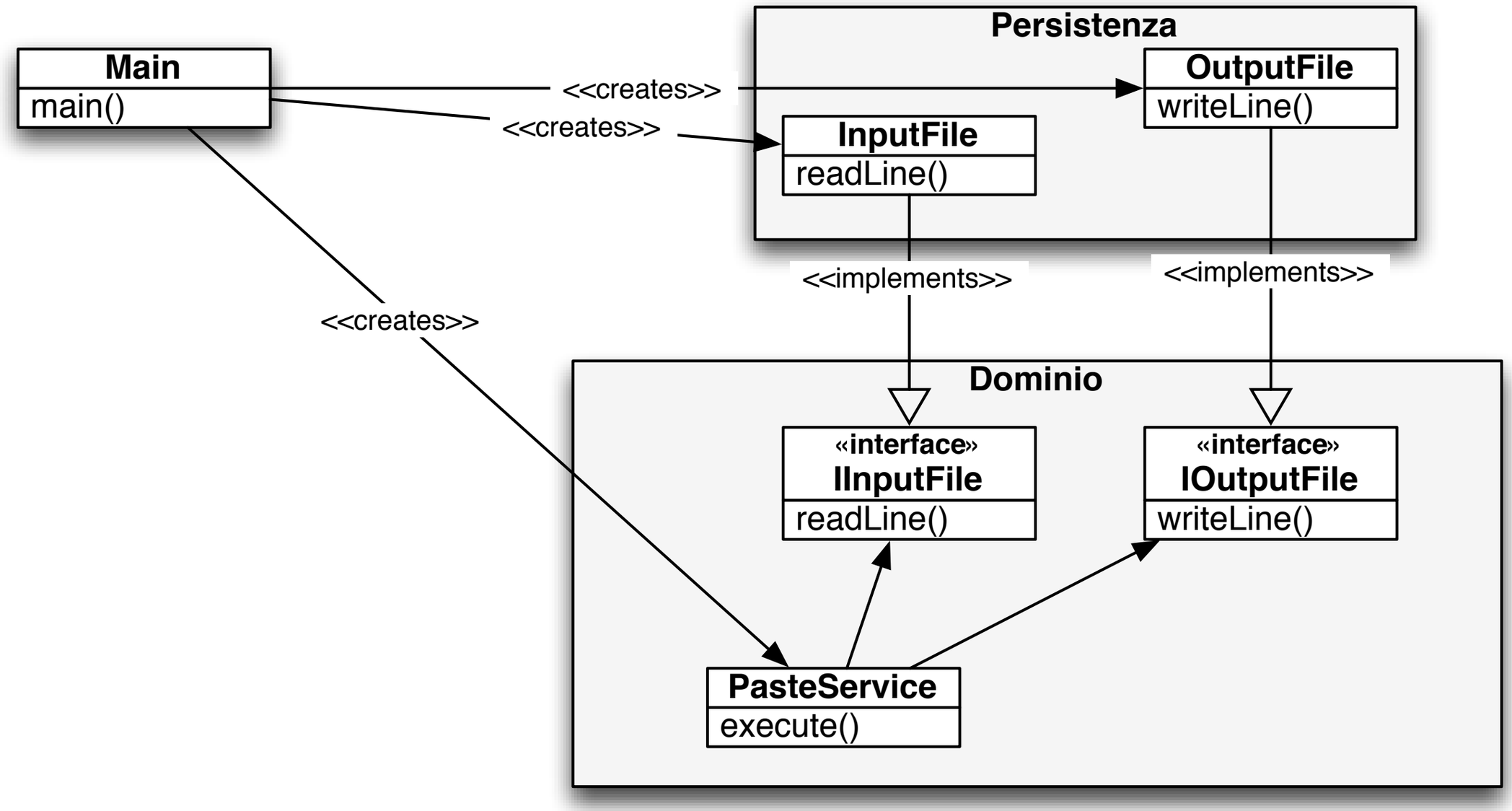
Non c’è molta struttura.

Primo tentativo di riprogettazione



Molto meglio. C'è però un problema: il PasteService invoca direttamente i metodi della persistenza. Il dominio dipende dal

Invertiamo le dipendenze



```
import java.io.IOException;

public class PasteService {

    public void execute(String firstFileName, String secondFileName, String outputFileName)
        throws IOException {

        InputFile first = new InputFile(firstFileName);
        InputFile second = new InputFile(secondFileName);
        OutputFile out = new OutputFile(outputFileName);

        String stringFromFirst, stringFromSecond;
        while ((stringFromFirst = first.readLine()) != null
                && (stringFromSecond = second.readLine()) != null) {
            out.writeLine(stringFromFirst + stringFromSecond);
        }

        first.close();
        second.close();
        out.close();
    }
}
```