An overview of Java™SE 7

Presenting some of the features introduced in Java™ Standard Edition 7

Presentation by: **Ricardo Fonseca**

Language improvements

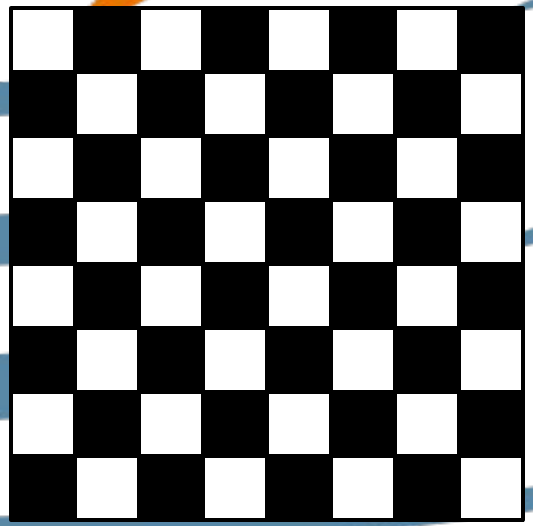
Language improvements

Binary Literals

```
int mask = 0b101010101010;

long aLong =
        0b1010000101000101101000010100010110100001010001011010000101000101L;

CHECKERBOARD = {
  0b0011001100110011,
  0b1100110011001100,
  0b0011001100110011,
  0b1100110011001100,
  0b0011001100110011,
  0b1100110011001100,
  0b0011001100110011,
  0b1100110011001100
};
```



Language improvements

Underscores in Numeric Literals

```
int mask = 0b1010_1010_1010;
long creditCardNumber = 1234_5678_9012_3456L;
float pi = 3.14_15F;
long hexWords = 0xCAFE_BABE;
long bytes = 0b11010010_01101001_10010100_10010010L;
```

Invalid

```
int pi2 = 3._1415F;
long big = 999_999_999_999_L;
int hex = 0_xFF;
```

Strings in switch Statements

```
public String getTypeOfDay (String dayOfWeek) {

    switch (dayOfWeek) {
        case "Monday“:      return "Start of work week";

        case "Tuesday":
        case "Wednesday":
        case "Thursday“:    return "Midweek";

        case "Friday“:      return "End of work week";

        case "Saturday":
        case "Sunday“:      return "Weekend“;

        default:            ...
    }
}
```

Type Inference for Generic Instance Creation

Also called the “diamond” operator

```
Map<String, List<String>> map =
             new HashMap<String, List<String>>();

// becomes

Map<String, List<String>> map = new HashMap<>();
```

The try-with-resources Statement

```
public static String readFirstLine (String path)
                                    throws IOException {
    try (BufferedReader br =
          new BufferedReader(new FileReader(path)))
    {
        return br.readLine();
    }
    // automatically closes the reader object
}
```

try-with-resources (behind the scenes)

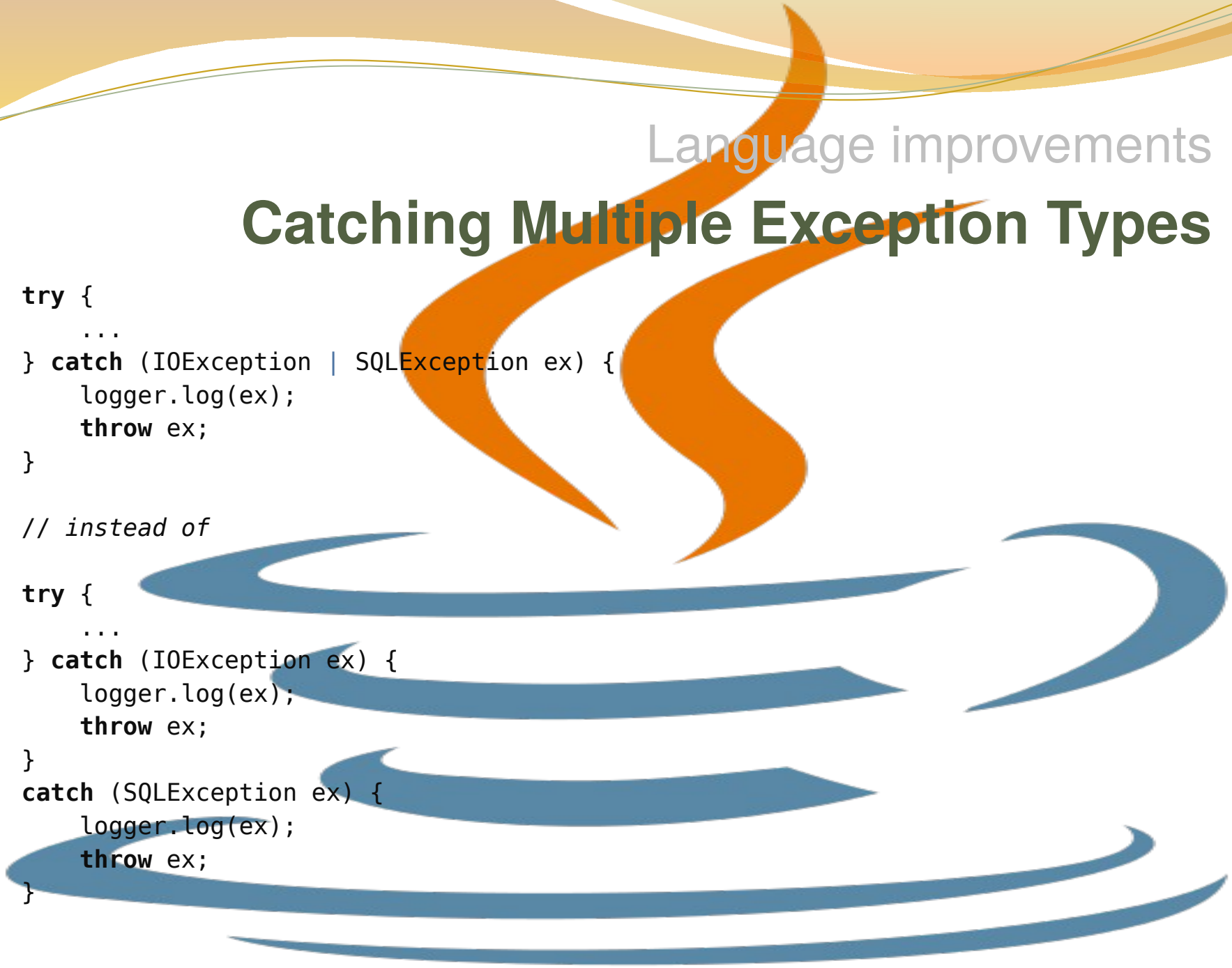
```
final BufferedReader br = new BufferedReader(new FileReader(path));
Throwable primaryEx = null;
try {
    return br.readLine();
} catch (Throwable e) {
    primaryEx = e;
    throw e;
} finally {
    if (br != null) {
        if (primaryEx != null) {
            try {
                br.close();
            } catch (Throwable suppressedEx) {
                primaryEx.addSuppressed(suppressedEx);
            }
        } else {
            br.close();
        }
    }
}
```

Catching Multiple Exception Types

```
try {
    ...
} catch (IOException | SQLException ex) {
    logger.log(ex);
    throw ex;
}

// instead of

try {
    ...
} catch (IOException ex) {
    logger.log(ex);
    throw ex;
}
catch (SQLException ex) {
    logger.log(ex);
    throw ex;
}
```



Rethrowing Exceptions with More Inclusive Type Checking

```
public void foo(String bar)
    throws FirstException, SecondException {

    try {
        // Code that may throw both
        // FirstException and SecondException
    }
    catch (Exception e) {
        throw e;
    }
}
```

Virtual Machine
(actually HotSpot™)
improvements

Garbage-First (G1) Collector

- A new type of concurrent garbage collector
  - Partitions heap into equal-sized regions
- Has predictable GC pause times
  - “Soft real-time”
- Performs quick compaction of memory heap
  - Compacts as it proceeds
  - Looks for regions with no live objects for immediate reclamation

Tiered Compilation

- Usually there is a “client” VM and a “server” VM
  - Client VM has faster start-up times
  - Server VM has better compilation optimizations during runtime
- Tiered compilation first compiles in client mode, then if the profiling indicates, it compiles with optimizations during runtime (server mode)

Compressed Oops

- An oop (ordinary object pointer) is normally the same size as a native machine pointer
  - 32 bits in a 32-bit architecture
  - 64 bits in a 64-bit architecture
- On a 64-bit machine, compressed oops are 32-bit values that must be scaled and added to a 64-bit base address in order to find the object they refer to

API improvements

API improvements

Fork Join Framework

```
solve(problem):
    if problem is small enough:
        solve problem directly
  (sequential algorithm)
    else:
        for part in subdivide(problem)
            fork subtask to solve part
    join all subtasks spawned in previous
  loop
    combine results from subtasks
```

Fork Join Framework

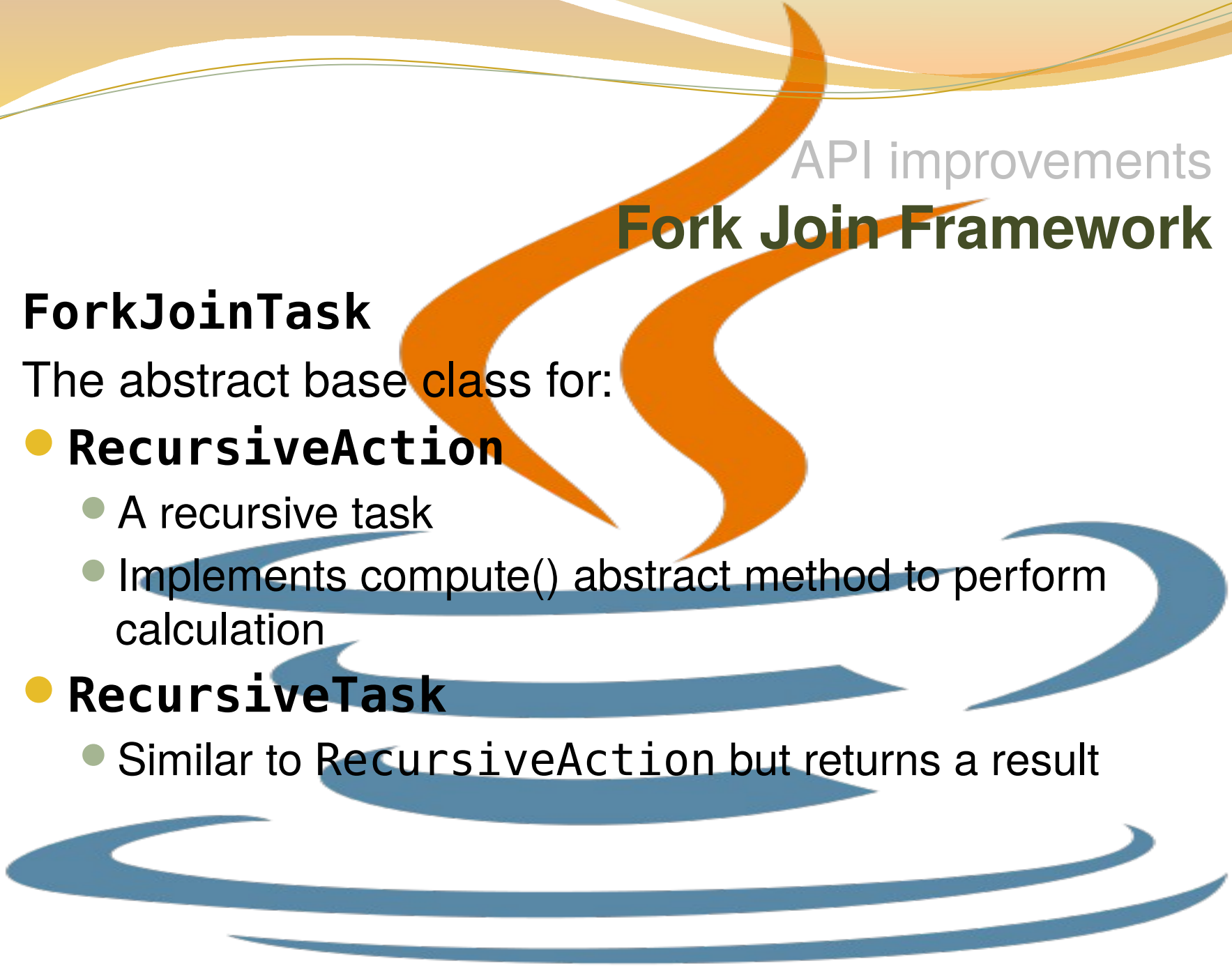
`ForkJoinPool`

Service for running `ForkJoinTasks`

- `pool.execute(aTask); // async`
- `pool.invoke(aTask); // wait`
- `pool.submit(aTask); // async + future`

Tasks are executed by threads created and managed by the pool

Fork Join Framework



`ForkJoinTask`

The abstract base class for:

- **`RecursiveAction`**
  - A recursive task
  - Implements compute() abstract method to perform calculation
- **`RecursiveTask`**
  - Similar to `RecursiveAction` but returns a result

API improvements

Fork Join Framework

Incrementing an array example

```
ForkJoinPool pool = new ForkJoinPool();
IncTask task = new IncTask(array, 0, array.length);
pool.invoke(task);
...

protected void compute() {
    if (hi - lo < THRESHOLD) {
        for (int i = lo; i < hi; i++)
            array[i]++;
    } else {
        int mid = (lo + hi) / 2;
        invokeAll(new IncTask(array, lo, mid),
                  new IncTask(array, mid, hi));
    }
}
```

java.nio

The central abstractions of the NIO APIs are:

- Buffers, which are containers for data
- Charsets and their associated decoders and encoders, which translate between bytes and Unicode characters
- Channels of various types, which represent connections to entities capable of performing I/O operations

java.nio.file

Two key navigation Helper Types:

- Class `java.nio.file.Paths`
  - Exclusively static methods to return a Path by converting a string or URI
- Interface `java.nio.file.Path`
  - Used for objects that represent the location of a file in a file system, typically system dependent

Typical use case:

- Use `Paths` to get a `Path`.
- Use `java.nio.Files` to do stuff.

API improvements

Path operations

```
// Make a reference to a File
Path src = Paths.get("/home/ricardo/readme.txt");
Path dst = Paths.get("/home/ricardo/copy_readme.txt");

// Make a reference to a directory
Path dir = Paths.get("/home/ricardo/");

//Navigation /home/ricardo -> /home/ricardo/tmp
Path tmpPath = dir.resolve("tmp");

//Create a relative path from dir -> ..
Path relativePath = tmpPath.relativize(dir);

// Convert to old File Format for your legacy apps
File file = aPathPath.toFile();
```

API improvements

Files operations

Class `java.nio.file.Files`

- Exclusively static methods to operate on files, directories and other types of files

`Files` helper class is feature rich:

- Copy
- Create Directories
- Create Links
- Use of system “temp” directory
- Delete
- Attributes - Modified/Owner/Permissions/Size, etc.
- Read/Write

API improvements

Directories

`DirectoryStream` iterate over entries

- Scales to large directories
- Uses less resources
- Implements `Iterable` and `Closeable`

Filtering support

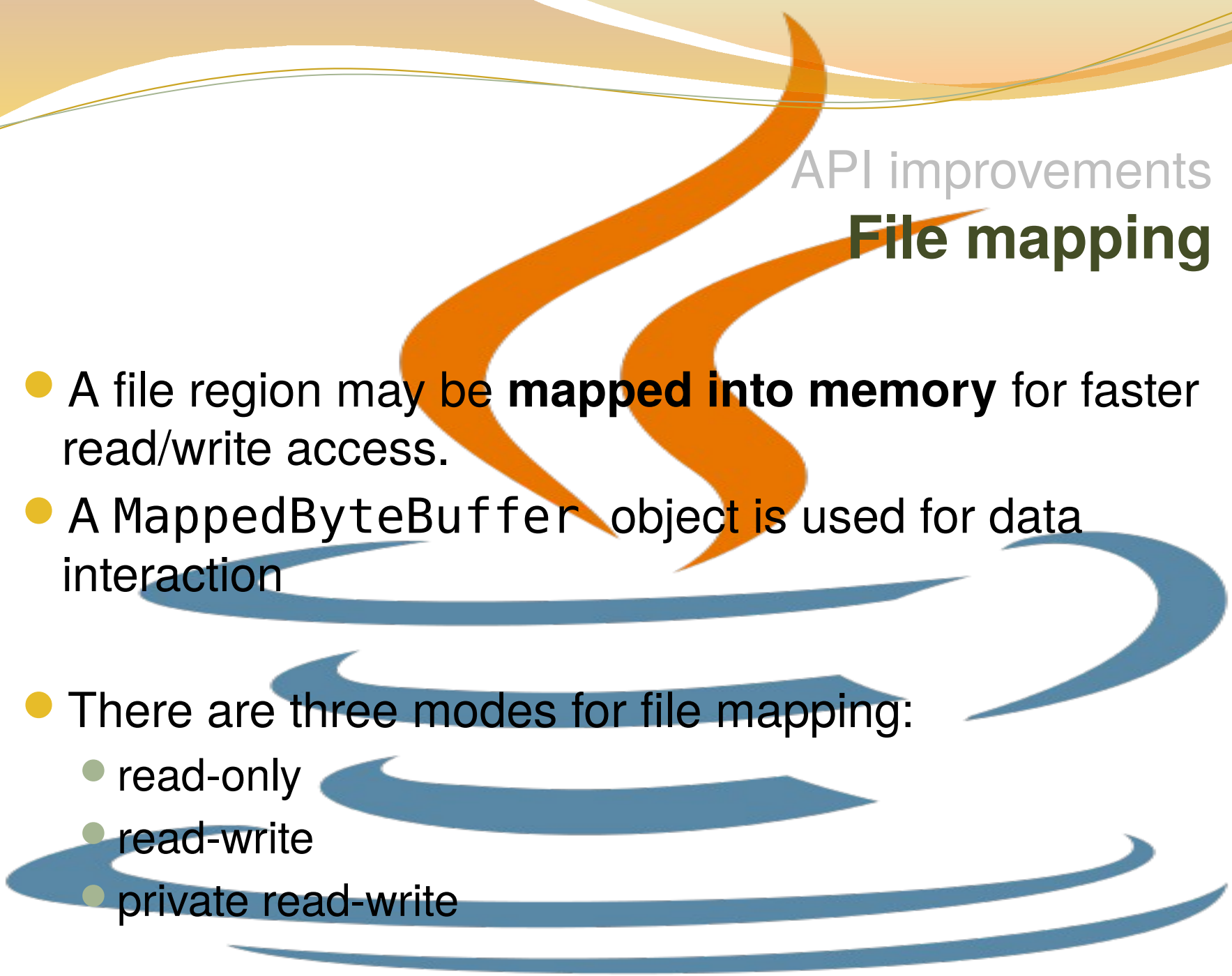
- Built-in support for glob, regex and custom filters

```
Path dir = Paths.get("/home/ricardo/src");
try (DirectoryStream<Path> ds =
  Files.newDirectoryStream(dir, "*.java")){
    for (Path file : ds)
        System.out.println(file.getName());
}
```

File locking

`java.nio.file.FileLock` class:

- A **region** of a file is locked. No two locks can overlap the same region (unless the lock is in shared mode, which is implementation specific).
- Locks are **advisory**, meaning other processes may not respect them

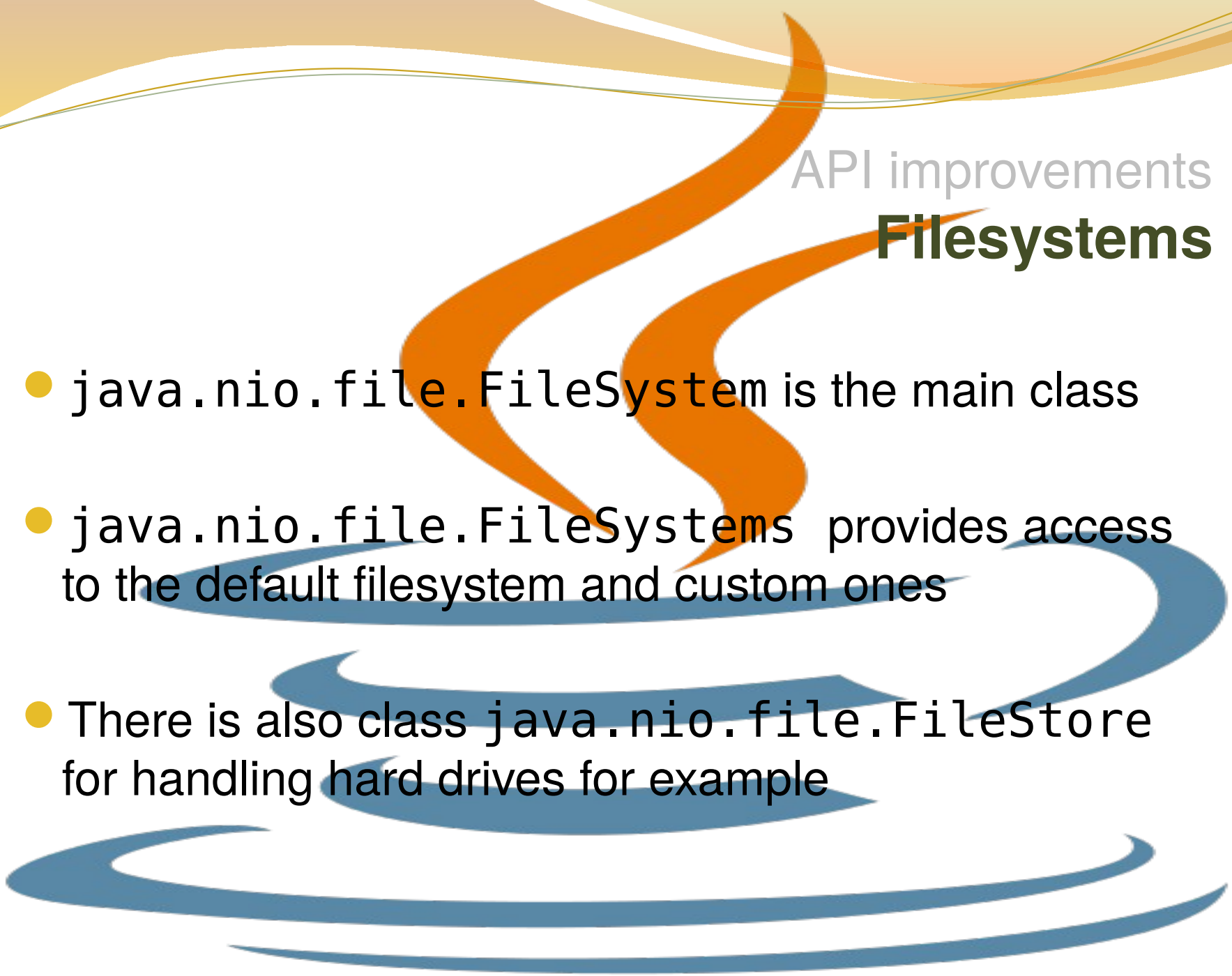


API improvements

File mapping

- A file region may be **mapped into memory** for faster read/write access.
- A `MappedByteBuffer` object is used for data interaction

- There are three modes for file mapping:
  - read-only
  - read-write
  - private read-write



API improvements

Filesystems

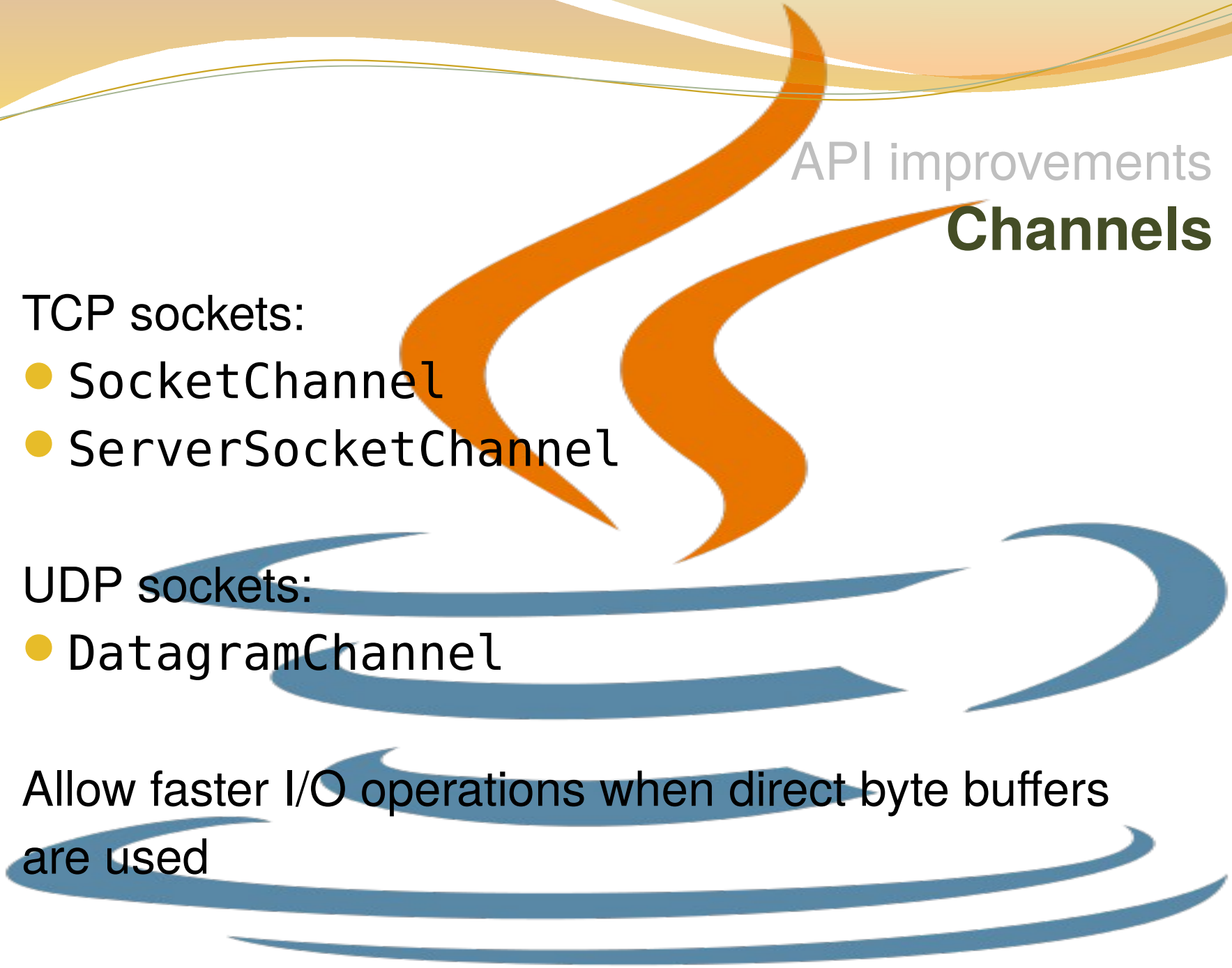
- `java.nio.file.FileSystem` is the main class
- `java.nio.file.FileSystems` provides access to the default filesystem and custom ones
- There is also class `java.nio.file.FileStore` for handling hard drives for example

A custom zip/jar filesystem

```
Path jarFile = Paths.get(“app.jar”);
FileSystem jarFS =
  FileSystems.newFileSystem(jarFile, null);

Path manif = jarFS.getPath(
                    “META-INF/MANIFEST.MF”);

// perform operations on file inside
// the jar
```



API improvements

Channels

TCP sockets:

- `SocketChannel`
- `ServerSocketChannel`

UDP sockets:

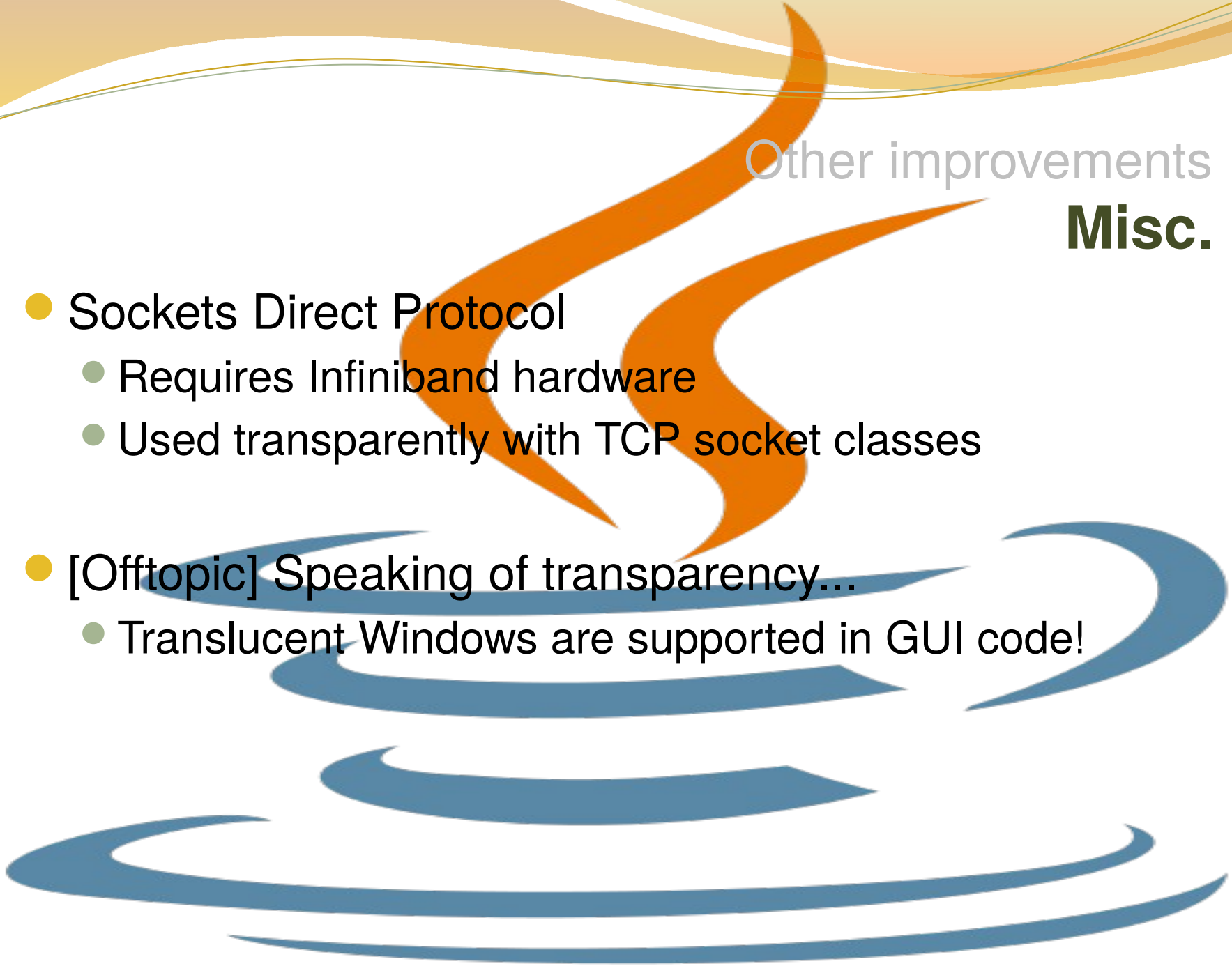
- `DatagramChannel`

Allow faster I/O operations when direct byte buffers are used

Other improvements

Misc.

- Sockets Direct Protocol
  - Requires Infiniband hardware
  - Used transparently with TCP socket classes
- [Offtopic] Speaking of transparency...
  - Translucent Windows are supported in GUI code!




Thank you!