# Exceptions

- Error Reporting
- Exception Return Path
- try/catch blocks
- finally
- Make your own Exception
- Runtime Exceptions

Introduction to Java

## See Also

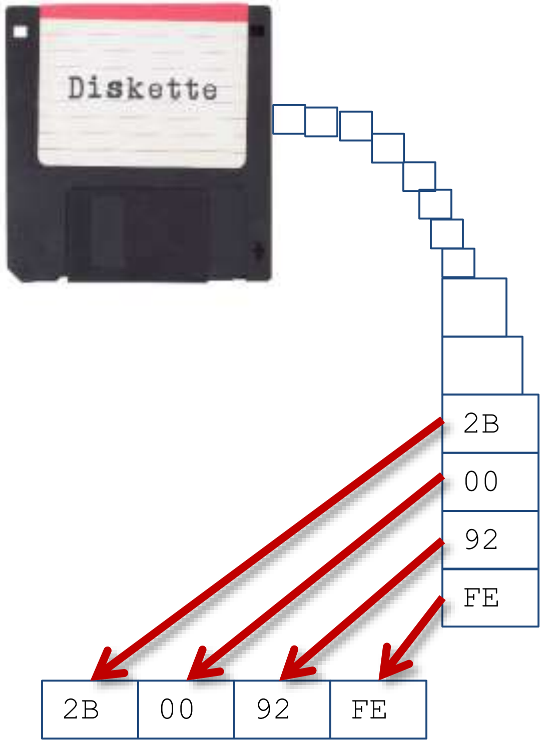
https://docs.oracle.com/javase/tutorial/essential/exceptions/

http://docs.oracle.com/javase/7/docs/api/java/lang/Exception.html

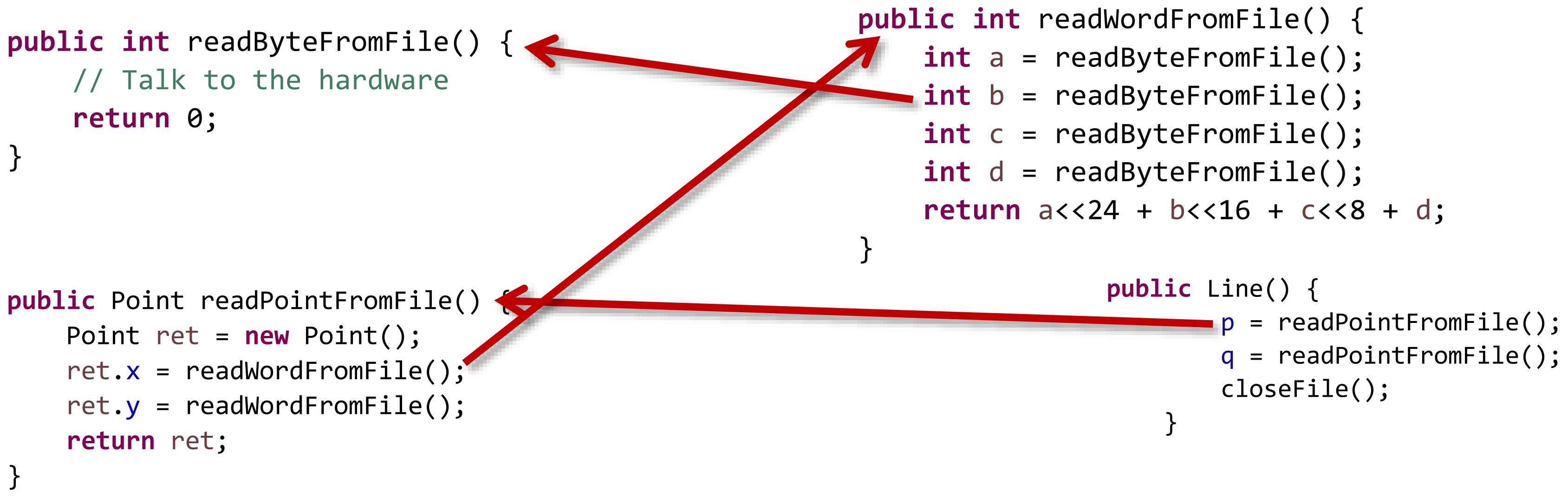
https://en.wikipedia.org/wiki/Exception_handling

# Reading a Line From Disk

```
public class Line {

    Point p;
    Point q;

    public void closeFile() {}

    public int readByteFromFile() {}

    public int readWordFromFile() {}

    public Point readPointFromFile() {}

    public Line() {}

    public static void main(String [] args) {
        Line n = new Line();
        System.out.println(n.p+" to "+n.q);
    }

}
```



# Error Checking



```
public int readByteFromFile() {
    // Talk to the hardware
    return 0;
}
```

```
public int readWordFromFile() {
    int a = readByteFromFile();
    int b = readByteFromFile();
    int c = readByteFromFile();
    int d = readByteFromFile();
    return a<<24 + b<<16 + c<<8 + d;
}
```

```
public Point readPointFromFile() {
    Point ret = new Point();
    ret.x = readWordFromFile();
    ret.y = readWordFromFile();
    return ret;
}
```

```
public Line() {
    p = readPointFromFile();
    q = readPointFromFile();
    closeFile();
}
```

- How do we handle errors?
- How do we report them back up the call stack?

## Merging in Error Conditions

```
public int readByteFromFile() {
    // Talk to the hardware
    //return 0;
    return -1; // Something bad happened
}
```

```
public int readWordFromFile() {
    int a = readByteFromFile();
    if(a<0) {
        return -1;
    }
    int b = readByteFromFile();
    if(b<0) {
        return -1;
    }
    int c = readByteFromFile();
    if(c<0) {
        return -1;
    }
    int d = readByteFromFile();
    if(d<0) {
        return -1;
    }
    return a<<24 + b<<16 + c<<8 + d;
}
```

- May not be a "special" value to return

# No Return

```
public Point readPointFromFile() {
     Point ret = new Point();
     ret.x = readWordFromFile();
     if(ret.x==-1) {
          return null;
     }
     ret.y = readWordFromFile();
     if(ret.y==-1) {
          return null;
     }
     return ret;
}
```

```
public Line() {
     p = readPointFromFile();
     if(p==null) {
          // ?
     }
     q = readPointFromFile();
     if(q==null) {
          // ?
     }
     closeFile();
}
```

```
public static void main(String [] args) {
     Line n = new Line();
     System.out.println(n.p+" to "+n.q);
}
```

- May not be able to return anything

## Two Return Paths

```
public int readByteFromFile() {
    // Talk to the hardware
    if(true) {
        // Something bad happened
        throw new Exception();
    }
    return 0;
}
```

- Use "throw" just like a "return" to abort the code
- The Exception object is returned to the caller instead of the normal return value
- You must declare that your method can do this

## Catching an Exception

```
public int readWordFromFile() {
    try {
        int a = readByteFromFile();
    } catch (Exception e) {
        // Do something
        throw e;
    }
    int b = readByteFromFile();
    int c = readByteFromFile();
    int d = readByteFromFile();
    return a<<24 + b<<16 + c<<8 + d;
}
```

- The “try” block of code tries to execute normally
- If an exception happens the CPU jumps to the “catch”
- Either way the CPU continues after the “catch” block

## Catching an Exception

```
public int readWordFromFile() throws Exception {
    try {
        int a = readByteFromFile();
        int b = readByteFromFile();
        int c = readByteFromFile();
        int d = readByteFromFile();
        return a<<24 + b<<16 + c<<8 + d;
    } catch (Exception e) {
        // Do something
        throw e;
    }
}
```

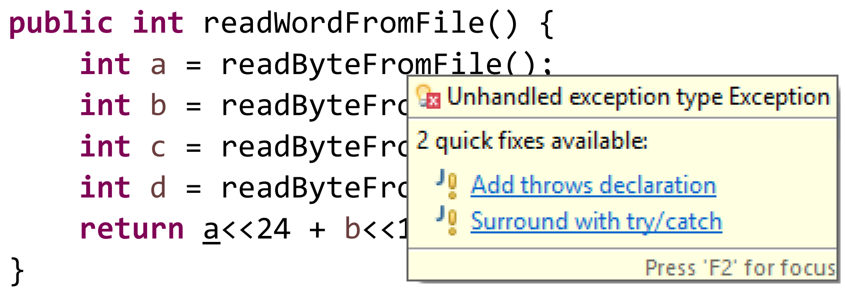
- You can put lots of instruction in the “try” block
- At any point an exception will abort to the “catch” block

## Bubbling up the Stack

```
public int readWordFromFile() throws Exception {
    int a = readByteFromFile();
    int b = readByteFromFile();
    int c = readByteFromFile();
    int d = readByteFromFile();
    return a<<24 + b<<16 + c<<8 + d;
}
```

- You can "handle" an exception by declaring your method throws it too
- Exceptions bubble back up the call stack to the first try/catch block they find

## You MUST Deal with Exceptions



```
public int readWordFromFile() {
    int a = readByteFromFile();
    int b = readByteFro
    int c = readByteFro
    int d = readByteFro
    return a<<24 + b<<1
}
```

Unhandled exception type Exception

2 quick fixes available:

- Add throws declaration
- Surround with try/catch

Press 'F2' for focus

- If you call a method that declares that it throws an exception then you must handle that exception in some way

# Looking at an Exception

```
public Line() throws Exception {
    p = readPointFromFile();
    q = readPointFromFile();
    closeFile();
}

public static void main(String [] args) {
    try {
        Line n = new Line();
        System.out.println(n.p+" to "+n.q);
    } catch (Exception e) {
        e.printStackTrace();
    }
}
```

- The Exception object keeps track of its path back up the stack

## Bubbling out of Main

```java
public Line() throws Exception {
    p = readPointFromFile();
    q = readPointFromFile();
    closeFile();
}

public static void main(String [] args) throws Exception {
    Line n = new Line();
    System.out.println(n.p+" to "+n.q);
}
```

- If an Exception bubbles out of main then the JVM calls printStackTrace on it for you

# Finally

```
public Line() throws Exception {
    p = readPointFromFile();
    q = readPointFromFile();
    closeFile();
}

public Line() throws Exception {
    try {
        p = readPointFromFile();
        q = readPointFromFile();
    } catch (Exception e) {
        closeFile();
        throw e;
    }
    closeFile();
}
```

- Be sure to close resources in BOTH paths: normal and exception

## Finally

```
public Line() throws Exception {
    try {
        p = readPointFromFile();
        q = readPointFromFile();
    } catch (Exception e) {
       throw e;
    } finally {
       closeFile();
   }
}
```

- The code in the “finally” block always executes

## Finally Woes

- Dealing with exceptions thrown from finally
- Dealing with “finally” on multiple resources
- Dealing with resources that never open

## Multiple Exceptions

```
public static void doLotsOfThings() throws IOException, NameException {
    // Code here
}

public static void doSomeThings() throws NotFoundException, NameException {
    // Code here
}

public static void main(String [] args) throws NameException {

    try {
        doLotsOfThings();
        doSomeThings();
    } catch (IOException e1) {
        // Code here
    } catch (NotFoundException e2) {
        // Code here
    }

}
```

- There are different kinds of exceptions
- You can try/catch many kinds

## Multiple Catches

```java
public static void doLotsOfThings() throws IOException, NameException {
    // Code here
}

public static void doSomeThings() throws NotFoundException, NameException {
    // Code here
}

public static void main(String [] args) throws NameException {

    try {
        doLotsOfThings();
        doSomeThings();
    } catch (IOException | NotFoundException e) {
        // Code here
    }

}
```

JAVA 7

- Use "or" to catch multiple kinds in a single block

# Your own Exception Class

```
public class MyException extends Exception {

    int temperature;
    int time;

}


    public static int checkThermometer() throws MyException {
        if(true) {
            MyException me = new MyException();
            me.temperature = -20;
            me.time = 1234;
            throw me;
        }
        return 20;
    }
```

- Extend the "Exception" class
- Add data and methods you need
- Exception has lots of constructors

## Catching by Inheritance

```
public static void main(String [] args) throws Exception {

    try {
        int val = checkThermometer();
    } catch (IOException e1) {
        // Code
    } catch (Exception e2) {
        // Code
    }

}
```

- Catches are checked in the order that you list them
- Exception is the base of most everything
- Catch generally … throw specifically

# Runtime Exceptions

```
public static void main(String [] args)
    throws NullPointerException,
        ArrayIndexOutOfBoundsException,
        ArithmeticException
{

    int [] dat = null;

    System.out.println(dat.length);

    dat[-1] = 20;

    int x = 5/0;

}
```

- RuntimeExceptions are for code problems
- They are not "checked" by the compiler
- You may catch these too but you rarely do

## Tinkering

- Make a static method that takes an array of integers and returns the index of the first "42" it finds.
- Should it return -1 if there is no "42"? Or should it throw an exception?
- Web search "python index method". Web search "java indexof method". The languages take different approaches. Which do you like?