Java Platform API Library Classes

Documentation and Using Library Classes

Zbigniew Dendzik

2026

Lecture Plan

- Java Platform – components
- Using library classes
- Java Platform API documentation
- The java.awt.Rectangle class
- Input/output handling
- Exception handling
- Text file operations
- External libraries

Java Platform – Components

The **Java Platform** consists of two main components:

- **Virtual Machine (JVM)** – bytecode interpreter
- **Class and interface library** – Java Platform API

The library provides ready-made classes for use in programs:

- Data structure handling
- Input/output operations
- Graphics and user interfaces
- Network communication

Java Platform API Specification

- Official documentation of all library classes
- Available online and as part of JDK installation
- Contains description of each class, field, constructor, and method
- Essential tool for Java programmers
- Must be able to use it efficiently

Documentation structure:

- General class description
- List of fields
- List of constructors
- List of methods with parameters and return types

Packages

- Library classes organized into packages
- Package = collection of related classes and interfaces
- Examples: `java.awt`, `java.io`, `java.util`
- Full class name: `package_name.ClassName`
- Example: `java.awt.Rectangle`

Importing classes:

- `import java.awt.Rectangle;` – import specific class
- `import java.awt.*;` – import all classes from package
- Allows using short class name

Example: java.awt.Rectangle Class

```
import java.awt.Rectangle;

public class Program
{
    public static void main(String[] args)
    {
        Rectangle obj = new Rectangle(0, 0, 4, 3);

        obj.translate(1, -1);

        System.out.println(obj);
    }
}
```

The java.awt.Rectangle Class

Represents a rectangle in two-dimensional space. **Fields:**

- `int x, y` – coordinates of upper-left vertex
- `int width, height` – width and height

Constructors:

- `Rectangle()` – default rectangle (0,0,0,0)
- `Rectangle(int x, int y, int w, int h)` – with parameters
- `Rectangle(Rectangle r)` – rectangle copy

Rectangle Class Methods (1)

Translation and transformation operations:

- `translate(int dx, int dy)` – shift by vector
- `setLocation(int x, int y)` – set position
- `setSize(int w, int h)` – set size
- `grow(int h, int v)` – enlarge/reduce

Rectangle Class Methods (2)

Methods testing relationships:

- `contains(int x, int y)` – is point inside?
- `contains(Rectangle r)` – does it contain rectangle?
- `intersects(Rectangle r)` – do rectangles intersect?
- `isEmpty()` – is rectangle empty?

Operations on rectangles:

- `intersection(Rectangle r)` – common part
- `union(Rectangle r)` – union (smallest rectangle containing both)

Example: Rectangle Intersection

```
Rectangle obj1 = new Rectangle(0, 0, 4, 3);
Rectangle obj2 = new Rectangle(1, 1, 4, 3);

Rectangle intersection = obj1.intersection(obj2);

System.out.println("Intersection: " + intersection);
// Result: java.awt.Rectangle[x=1,y=1,width=3,height=2]
```

Example: Checking Containment

```
Rectangle obj1 = new Rectangle(1, 1, 4, 5);
Rectangle obj2 = new Rectangle(2, 0, 2, 3);

boolean contains = obj2.contains(obj1);

System.out.println("Does obj2 contain obj1? " + contains);
// Result: false
```

Example: Checking Point

```
Rectangle obj = new Rectangle(-3, 0, 6, 3);

boolean inside = obj.contains(2, -1);

System.out.println("Is point (2,-1) inside rectangle? "
                   + inside);
// Result: false (point outside rectangle)
```

Example: Rectangle Intersection Test

```
Rectangle obj1 = new Rectangle(1, 1, 4, 5);
Rectangle obj2 = new Rectangle(4, -3, 4, 3);

boolean intersects = obj1.intersects(obj2);

System.out.println("Do rectangles intersect? "
                   + intersects);
// Result: false
```

Exception Handling in Java

Exception – exceptional situation during program execution. **Types of exceptions:**

- `IOException` – input/output operation errors
- `NumberFormatException` – invalid number format
- `FileNotFoundException` – file does not exist
- `Exception` – general exception class

Exception handling:

- `try` block – code that may throw exception
- `catch` block – handling specific exception
- Multiple `catch` blocks possible

try-catch Syntax

```
try
{
    // Code that may throw exception
    // e.g., input/output operations
}
catch(IOException e)
{
    // Handle IOException
    System.out.println("I/O operation error");
}
catch(NumberFormatException e)
{
    // Handle NumberFormatException
    System.out.println("Invalid number format");
}
```

Keyboard Input – Classes

Classes for reading

- `System.in` – standard input (keyboard)
- `InputStreamReader` – byte to character conversion
- `BufferedReader` – buffered text line reading

Typical construction:

- `BufferedReader br = new BufferedReader(`
- `    new InputStreamReader(System.in));`
- Method `readLine()` returns String
- Requires `IOException` handling

Example: Keyboard Input

```
import java.io.*;

public class A
{
    static double RATE = 3.8;

    public static void main(String[] args)
    {
        try
        {
            BufferedReader br = new BufferedReader(
                new InputStreamReader(System.in));

            System.out.print("$: ");
            String str = br.readLine();

            double x = Double.parseDouble(str);
            System.out.println("PLN: " + x * RATE);
```

Example: Keyboard Input (cont.)

```
        catch(IOException e1)
        {
            System.out.println("I/O operation exception");
        }
        catch(NumberFormatException e2)
        {
            System.out.println("invalid number format");
        }
    }
}
```

String to Number Conversion

Parsing methods:

- `Double.parseDouble(String s)` – String → double
- `Integer.parseInt(String s)` – String → int
- `Float.parseFloat(String s)` – String → float
- `Long.parseLong(String s)` – String → long

Error handling:

- Throw `NumberFormatException` on invalid format
- Should be used in try-catch block
- Example invalid `"abc"`, `"12.34.56"`

File Operations – Classes

Writing to file:

- `FileWriter` – writing characters to file
- `BufferedWriter` – buffered writing
- `PrintWriter` – convenient println(), print() methods

Reading from file:

- `FileReader` – reading characters from file
- `BufferedReader` – buffered reading with readLine() method

Note:

- Operations may throw `IOException`
- Must close streams after completion: `close()`

Example: Writing to File

```
import java.io.*;

BufferedReader br = new BufferedReader(
    new InputStreamReader(System.in));

try
{
    System.out.print("first name: ");
    String firstName = br.readLine();
    System.out.print("last name: ");
    String lastName = br.readLine();

    PrintWriter file = new PrintWriter(
        new BufferedWriter(
            new FileWriter("file.txt", true)));

    file.println(firstName + " " + lastName);
    file.close();
```

Example: Writing to File (cont.)

```
catch(Exception e)
{
    System.out.println(e);
}
```

Parameter `true` in FileWriter:

- `new FileWriter("file.txt", true)` – appending
- `new FileWriter("file.txt", false)` – overwriting
- Default: `false`

Example: Reading from File

```
try
{
    BufferedReader file = new BufferedReader(
        new FileReader("file.txt"));

    String str;

    while(file.ready())
    {
        str = file.readLine();
        System.out.println(str);
    }

    file.close();
}
catch(Exception e)
{
    System.out.println(e);
```

BufferedReader Class Methods

Reading from file:

- `readLine()` – reads one line as String
- `ready()` – checks if data available to read
- `close()` – closes stream
- `read()` – reads single character

Typical usage pattern:

- Loop: `while(file.ready())`
- In loop: `str = file.readLine();`
- After completion: `file.close();`

save() Method in Class

Task: Rectangle class with method saving to file.

```
class Rectangle
{
    double length;
    double width;

    void save(String fileName)
    {
        try
        {
            PrintWriter file = new PrintWriter(
                new FileWriter(fileName));

            file.println("length: " + length);
            file.println("width: " + width);
            file.close();
        }
```

save() Method in Class (cont.)

```
        catch(Exception e)
        {
            System.out.println("Write error: " + e);
        }
    }
}
```

Usage:

```
Rectangle obj = new Rectangle(5.0, 3.0);
obj.save("rectangle.txt");
```

External Libraries

- Java allows using libraries outside standard API
- Open source and commercial libraries
- Extend platform capabilities
- Example applications:
  - Cryptography (BouncyCastle)
  - Mathematical analysis (Apache Commons Math)
  - File format handling (Apache POI, iText)
  - Network communication (Netty)

Installation:

- Download library JAR file
- Add to classpath during compilation and execution

Example: Provider Registration

```
import java.security.*;
import org.bouncycastle.jce.provider.BouncyCastleProvider;

public class RSATest
{
    public static void main(String args[]) throws Exception
    {
        System.out.println("Registering provider...");

        Provider prov = new BouncyCastleProvider();
        Security.addProvider(prov);

        System.out.println("Provider registered!");
    }
}
```

RSA Encryption – Concept

RSA – asymmetric encryption algorithm. **Key pair:**

- **Public key** – for encryption, can be shared
- **Private key** – for decryption, confidential

Process:

- Sender encrypts message with recipient's public key
- Recipient decrypts with private key
- No one else can read the message

Example: Generating RSA Keys

```
KeyPairGenerator kpgen =
    KeyPairGenerator.getInstance("RSA", "BC");

kpgen.initialize(1024, new SecureRandom());

KeyPair pair = kpgen.genKeyPair();

PrivateKey priv = pair.getPrivate();
PublicKey pub = pair.getPublic();

System.out.println("Keys generated!");
```

Summary

- **Java Platform API** – rich class library
- **Documentation** – essential programmer tool
- **Packages** – class organization, import
- **java.awt.Rectangle** – library class example
- **Exception handling** – try-catch
- **Input/output** – BufferedReader, PrintWriter
- **Text files** – reading and writing
- **External libraries** – extending capabilities