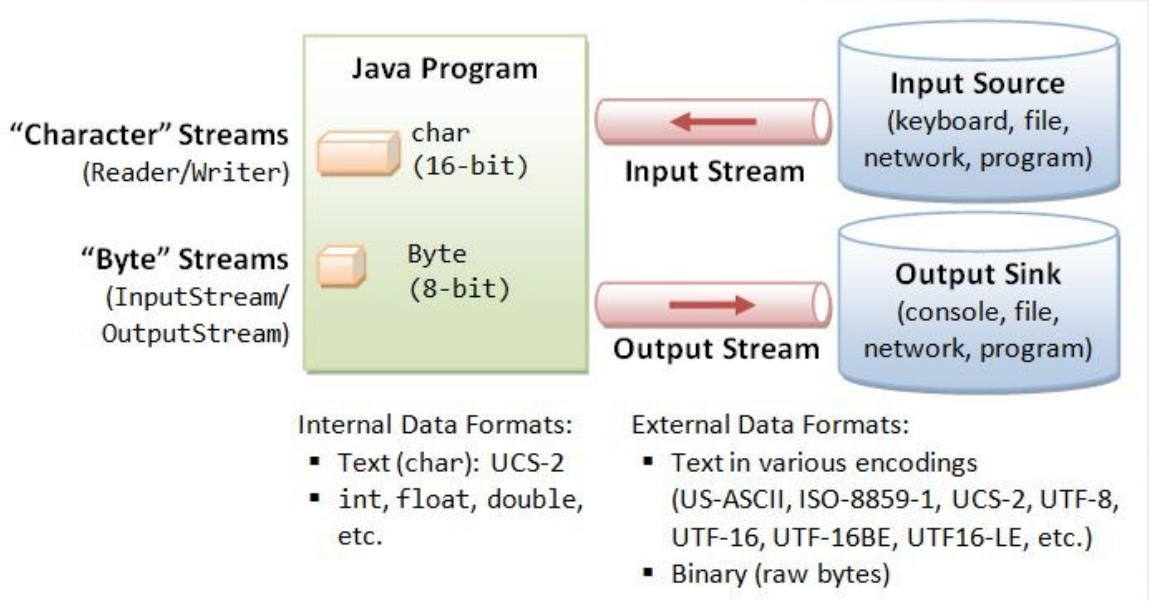
Additional Slides

- Java File Input & Output
- SOLID Principles

Java I/O Patterns



Java Input and Output (I/O)

- Scanner, using Command Line and File

```
Scanner scanner = new Scanner(System.in);
scanner.nextLine();     // read from command line

scanner = new Scanner(new FileInputStream("file.txt"));
scanner.nextLine();     // read from file
```

Java I/O: FileInputStream

- Fine-grained control for low-level reading
- Read bytes from files
- Control over size of the buffer
- Control over each byte
- Useful for custom file formats

```
InputStream inputStream = new FileInputStream("input.txt");
byte[] buffer = new byte[1000];
int n = inputStream.read(buffer, 0, 10);

if (n == -1) {
    // End of file
    System.exit(0);
}

for (int i = 0; i < n; i++) {
    System.out.print((char) buffer[i]);
}
```

Exercise

- Use FileInputStream to read from a file.
- Explain what BufferedInputStream does.
- Find the difference between InputStream and Reader for reading from a source.
- Trying switching FileInputStream with System.in.
- How much change is required in the program?

Java I/O: Writing to a file

- Define an OutputStream
- Connect it to a “writer”
  - PrintWriter
  - FileWriter
- PrintWriter is suitable for writing large texts such as lines of Strings
- FileWriter provides character-based writing, using a buffer

```
OutputStream outputStream = new FileOutputStream("out.txt");
PrintWriter writer = new PrintWriter(outputStream);

for (int i = 0; i < 3; i++) {
    String line = "Sample Text ".repeat(i + 1);
    writer.println(line);
}

writer.close();
```

SOLID Principles

S – Single-responsibility Principle

O – Open-closed Principle

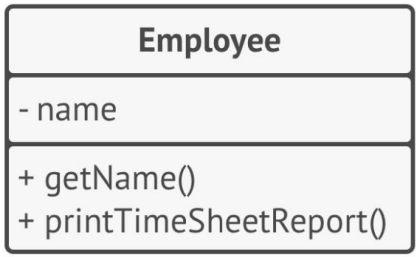
L – Liskov Substitution Principle

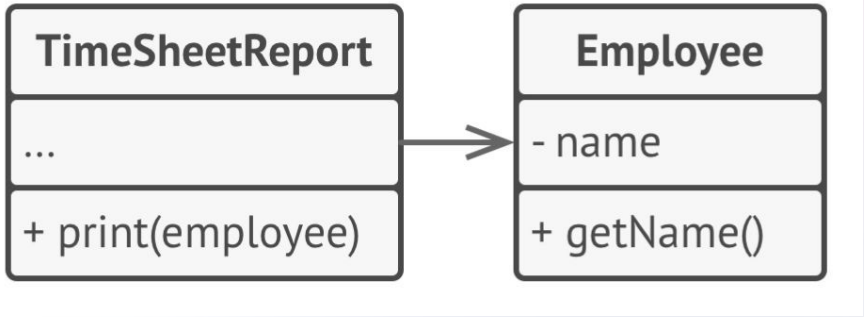
I – Interface Segregation Principle

D – Dependency Inversion Principle

Single-responsibility Principle

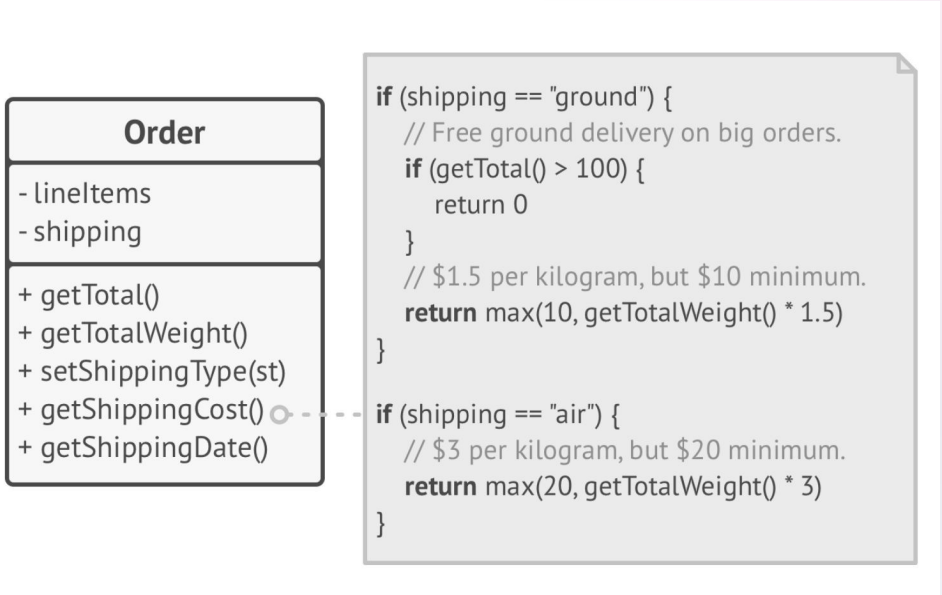
- A class should have just one reason to change





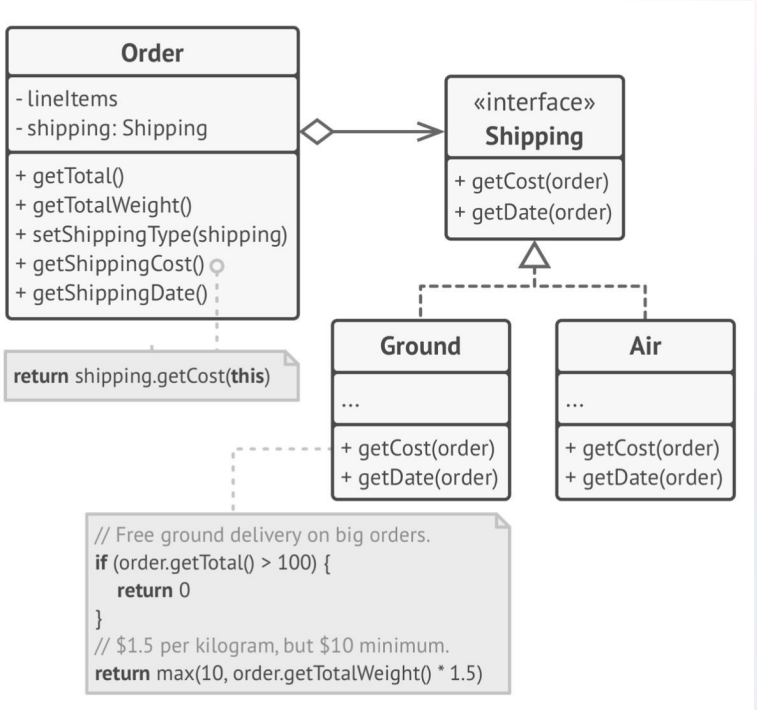
Open-closed Principle

- Classes should be open for **extension** but closed for **modification**.



Open-closed Principle

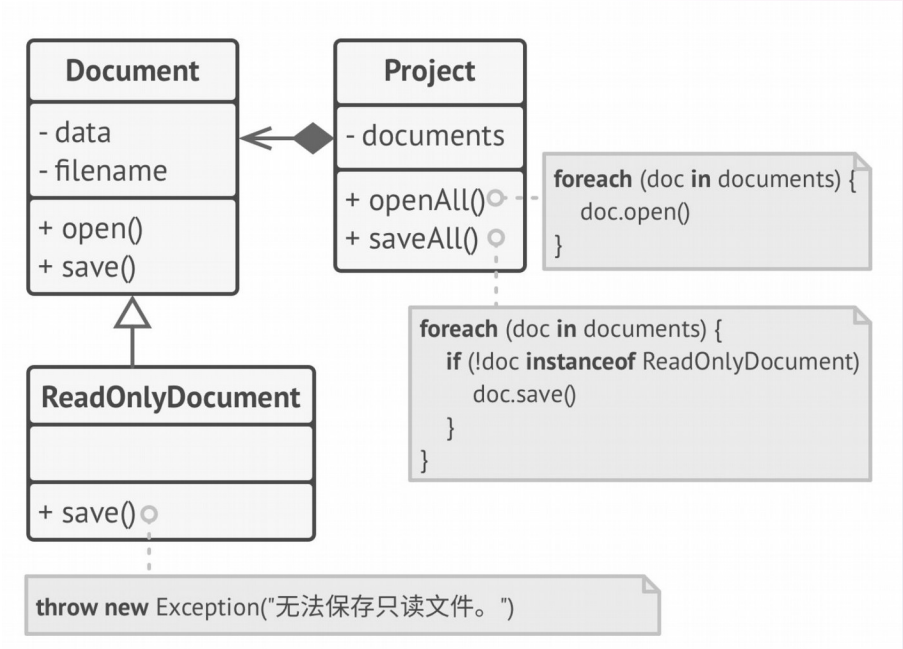
- Classes should be open for extension but closed for modification.



sourcemind

Liskov Substitution Principle

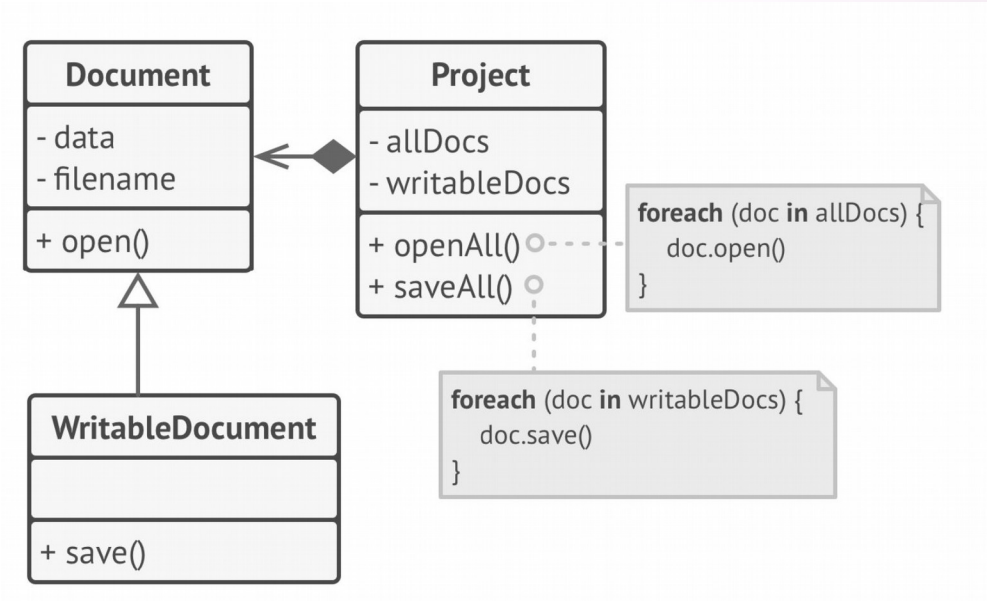
- When extending a class, remember that you should be able to pass objects of the subc
  in place of objects of the parent class without breaking the client code.



sourcemind

Liskov Substitution Principle

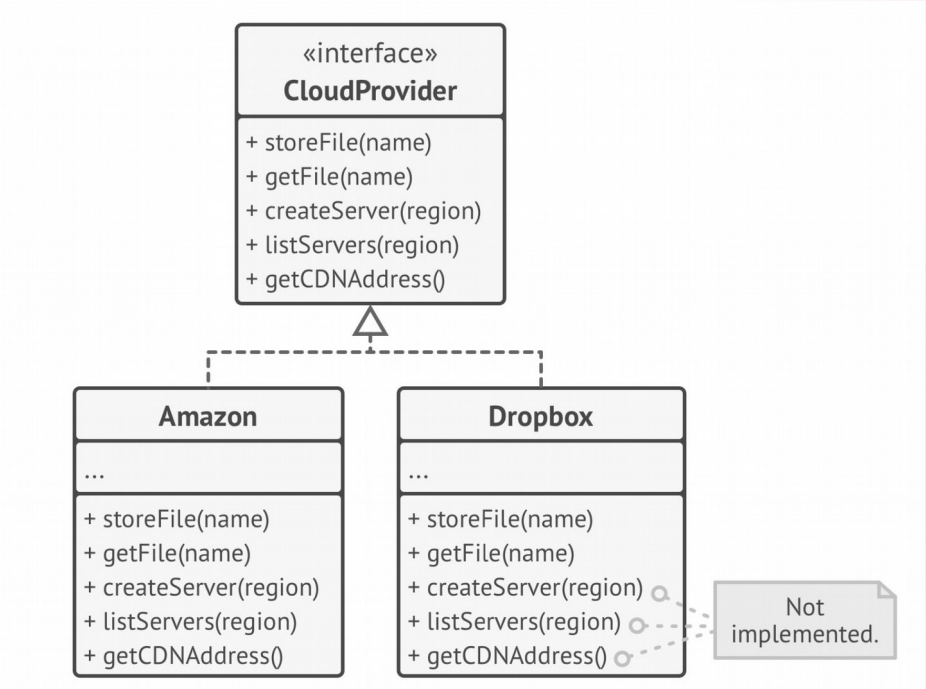
- When extending a class, remember that you should be able to pass objects of the subc
  in place of objects of the parent class without breaking the client code.



sourcemind

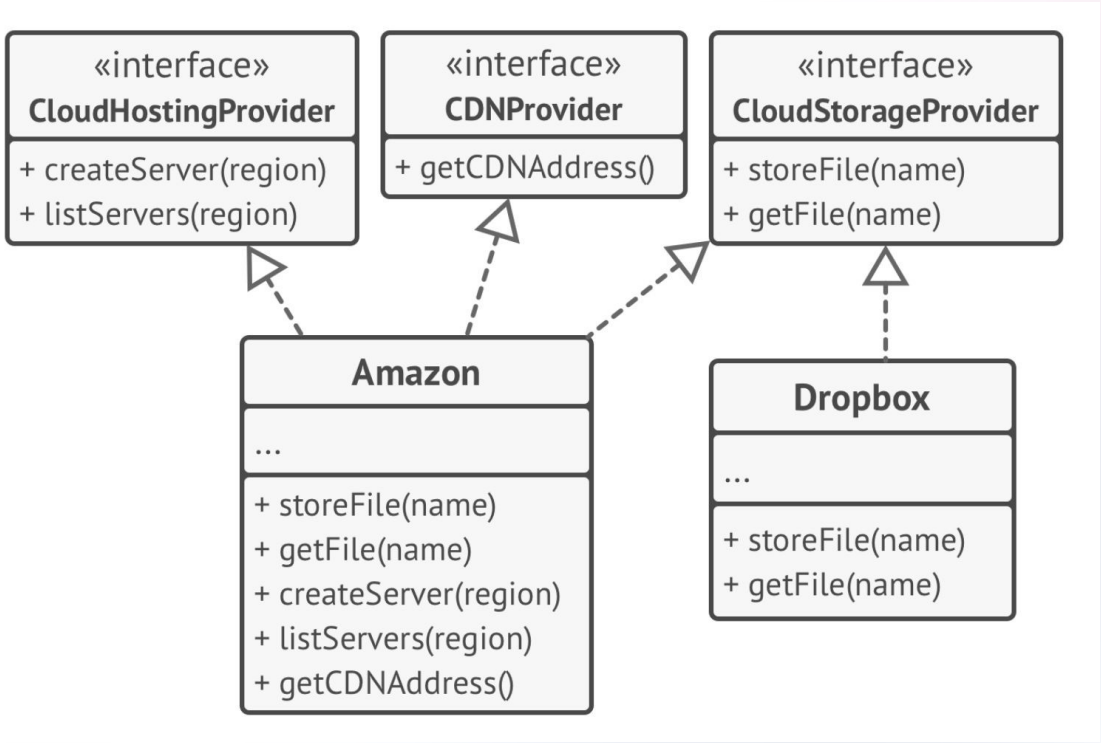
Interface Segregation Principle

- Clients should not be forced to depend on methods they do not use.



Interface Segregation Principle

- Clients should not be forced to depend on methods they do not use.



Dependency Inversion Principle

- High-level classes should not depend on low-level classes. Both should depend on abstraction.
- Abstractions should not depend on details.
- Details should depend on abstractions.