既支持抽象类，又支持接口，所以我们使用 Java 进行举例讲解，以便读者对这两个语法概念有直观的认识。

首先，我们看一下如何在 Java 中定义抽象类。

下面这段代码是一个典型的抽象类使用场景（模板设计模式）。Logger 是一个记录日志的抽象类，FileLogger 类和 MessageQueueLogger 类继承 Logger 类，分别实现不同的日志记录方式：将日志输出到文件中和将日志输出到消息队列中。FileLogger 和 MessageQueueLogger 两个子类复用了父类 Logger 中的 name、enabled、minPermittedLevel 属性，以及 log() 方法，但因为这两个子类输出日志的方式不同，所以它们又各自重写了父类中的 doLog() 方法。

```
public abstract class Logger {
  private String name;
  private boolean enabled;
  private Level minPermittedLevel;

  public Logger(String name, boolean enabled, Level minPermittedLevel) {
    this.name = name;
    this.enabled = enabled;
    this.minPermittedLevel = minPermittedLevel;
  }

  public void log(Level level, String message) {
    boolean loggable = enabled && (minPermittedLevel.intValue() <= level.intValue());
    if (!loggable) return;
    doLog(level, message);
  }

  protected abstract void doLog(Level level, String message);
}

//抽象类的子类：输出日志到文件
public class FileLogger extends Logger {
  private Writer fileWriter;

  public FileLogger(String name, boolean enabled,
    Level minPermittedLevel, String filepath) {
    super(name, enabled, minPermittedLevel);
    this.fileWriter = new FileWriter(filepath);
  }

  @Override
  public void doLog(Level level, String mesage) {
    //格式化level和message，并输出到日志文件
    fileWriter.write(...);
  }
}

//抽象类的子类：输出日志到消息中间件（如Kafka）
public class MessageQueueLogger extends Logger {
  private MessageQueueClient msgQueueClient;

  public MessageQueueLogger(String name, boolean enabled,
    Level minPermittedLevel, MessageQueueClient msgQueueClient) {
    super(name, enabled, minPermittedLevel);
    this.msgQueueClient = msgQueueClient;
  }

  @Override
```