什么是IoC（Inversion of Control）、 DIP（Dependency Inversion Principle）、Dependency Injection ？请举例说明实现方式

答：**IoC（Inversion of Control，控制反转）**：这是一种设计原则，它将对象的创建和依赖关系的管理从代码中转移到外部容器。在传统编程中，对象的创建和依赖关系的维护由程序自身负责，而在 IoC 中，控制权被反转，交给了外部容器。这样做的好处是提高了代码的可维护性和可测试性；**DIP（Dependency Inversion Principle，依赖倒置原则）**：它是面向对象设计中的一个重要原则，核心思想是高层模块不应该依赖低层模块，二者都应该依赖抽象；抽象不应该依赖细节，细节应该依赖抽象。这有助于降低模块间的耦合度，提高系统的灵活性和可扩展性；**Dependency Injection（依赖注入）**：它是实现 IoC 的一种具体方式，通过将对象的依赖项通过构造函数、方法参数或者属性注入到对象中，而不是在对象内部直接创建依赖项。

Code：

// 定义一个抽象接口，体现 DIP 中的依赖抽象  
interface MessageService {  
 void sendMessage(String message);  
}  
  
// 具体的消息服务实现类，细节依赖抽象  
class EmailService implements MessageService {  
 @Override  
 public void sendMessage(String message) {  
 System.*out*.println("Sending email: " + message);  
 }  
}  
  
class SMSService implements MessageService {  
 @Override  
 public void sendMessage(String message) {  
 System.*out*.println("Sending SMS: " + message);  
 }  
}  
  
// 高层模块，依赖抽象而不是具体实现  
class Notification {  
 private MessageService messageService;  
  
 // 通过构造函数进行依赖注入，体现 Dependency Injection  
 public Notification(MessageService messageService) {  
 this.messageService = messageService;  
 }  
  
 public void sendNotification(String message) {  
 messageService.sendMessage(message);  
 }  
}  
  
// 测试类  
public class Main {  
 public static void main(String[] args) {  
 // 创建具体的消息服务实例  
 MessageService emailService = new EmailService();  
 MessageService smsService = new SMSService();  
  
 // 通过依赖注入将消息服务注入到 Notification 类中，体现 IoC  
 Notification emailNotification = new Notification(emailService);  
 Notification smsNotification = new Notification(smsService);  
  
 // 发送通知  
 emailNotification.sendNotification("This is an email notification.");  
 smsNotification.sendNotification("This is an SMS notification.");  
 }  
}

运行结果：

