1. O Princípio da Inversão de Dependência (DIP) sugere que:  
   A) Módulos de alto nível devem depender diretamente de módulos de baixo nível.  
   B) Módulos não devem usar interfaces ou abstrações.  
   C) Módulos de alto nível e baixo nível devem depender de abstrações.  
   D) O software deve ser desenvolvido sem nenhum planejamento.  
     
   2) Para implementar o DIP em Java, devemos usar:  
   A) Comandos de impressão direta.  
   B) Interfaces ou classes abstratas.  
   C) Apenas variáveis globais.  
   D) Laços de repetição infinitos.  
     
   3) Aplicar o DIP resulta em:  
   A) Um aumento no acoplamento entre módulos.  
   B) Uma redução no acoplamento entre módulos.   
   C) Uma impossibilidade de testar o software.  
   D) Uma necessidade de reescrever o código diariamente.  
     
   4) Qual é uma vantagem do uso do DIP?  
   A) Torna o código mais difícil de entender.  
   B) Aumenta a dependência em implementações específicas.  
   C) Permite a substituição de módulos sem grandes impactos no sistema.   
   D) Exige que todo o código seja escrito sem funções.  
     
     
   5) Explique como a implementação do Princípio da Inversão de Dependência pode beneficiar o processo de testes em um projeto de software. Inclua um exemplo prático baseado em Java para ilustrar sua explicação.

A implementação do Princípio da Inversão de Dependência (DIP) pode beneficiar significativamente o processo de testes em um projeto de software, principalmente por meio da facilitação da criação de testes unitários e da realização de testes de integração de forma mais eficiente. Isso ocorre porque o DIP promove a redução do acoplamento entre os módulos do sistema, permitindo a substituição de implementações específicas por abstrações, o que facilita a realização de testes isolados e a introdução de mocks ou stubs para simular comportamentos específicos.

public interface UserRepository {

User getUserById(int userId);

void saveUser(User user);

}

public class UserService {

private final UserRepository userRepository;

public UserService(UserRepository userRepository) {

this.userRepository = userRepository;

}

public void createUser(String username, String email) {

User newUser = new User(username, email);

userRepository.saveUser(newUser);

}

public User getUser(int userId) {

return userRepository.getUserById(userId);

}

}

public class DatabaseUserRepository implements UserRepository {

}

public class MockUserRepository implements UserRepository {

private Map<Integer, User> users = new HashMap<>();

@Override

public User getUserById(int userId) {

return users.get(userId);

}

@Override

public void saveUser(User user) {

users.put(user.getId(), user);

}

}

public class User {

private int id;

private String username;

private String email;

// Construtor, getters e setters

// Outros métodos relevantes

}