Принципы ООП с примерами кода Java

1. Наследование — способность объекта или класса базироваться на другом объекте или классе. Это главный механизм для повторного использования кода. Наследственное отношение классов четко определяет их иерархию;

Пример в Java:

public class Parent {  
 public void parentFunction\_1(){  
 System.*out*.println("\texecute first function");  
 }  
 public void parentFunction\_2(){  
 System.*out*.println("\texecute second function");  
 }  
 public void parentFunction\_3(){  
 System.*out*.println("\texecute third function");  
 }  
}

package OOP;  
  
public class Child extends Parent{  
 public void childFunction(){  
 System.*out*.println("Start childFunction work");  
 parentFunction\_1();  
 parentFunction\_2();  
 parentFunction\_3();  
 System.*out*.println("Finish childFunction work");  
 }  
}

public class Examples {  
  
 public static void main(String[] args){  
 Child child = new Child();  
 child.childFunction();  
 }  
}

Вывод:

Start childFunction work

execute first function

execute second function

execute third function

Finish childFunction work

Process finished with exit code 0

1. Инкапсуляция в это метод, при котором состояние объекта скрыто от внешнего мира, а набор общедоступных методов для доступа к этому состоянию становится доступным.

Пример в Java:

public class Encapsulation {  
 //Защищаемые данные  
  
 private String safe\_data = "Password:1234";  
 private String password = "1234";  
  
 //Публичный интерфейс для получения доступа к данным  
  
 public String getSafe\_data(String password) {  
 //Проверка для доступа к данным  
 if (password.equals(this.password)){  
 return safe\_data;  
 } else {  
 return "No access to data";  
 }  
 }  
  
 public boolean setSafe\_data(String password, String safe\_data) {  
 //Проверка для доступа к данным  
 if (password.equals(this.password)){  
 this.safe\_data = "Password:" + safe\_data;  
 this.password = safe\_data;  
 return true;  
 } else {  
 return false;  
 }  
 }  
}

1. Абстракция - это придание объекту характеристик, которые чётко определяют его концептуальные границы, отличая от всех других объектов. Основная идея состоит в том, чтобы отделить способ использования составных объектов данных от деталей их реализации в виде более простых объектов, подобно тому, как функциональная абстракция разделяет способ использования функции и деталей её реализации в терминах более примитивных функций, таким образом, данные обрабатываются функцией высокого уровня с помощью вызова функций низкого уровня.

Пример в Java:

public class Abstract {  
 private void smallFunction\_1(){  
 // малая функция 1  
 System.*out*.println("Выполняется первая часть работы");  
 }  
  
 private void smallFunction\_2(){  
 // малая функция 2  
 System.*out*.println("Выполняется вторая часть работы");  
 }  
  
 private void smallFunction\_3(){  
 // малая функция 3  
 System.*out*.println("Выполняется третья часть работы");  
 }  
  
 public void mainFunction(){  
 // Основная функция доступная пользователю  
 // Для упрощения написания кода функция разбита на подфункции  
 smallFunction\_1();  
 smallFunction\_2();  
 smallFunction\_3();  
 System.*out*.println("Финальный результат");  
   
 }  
}

1. Полиморфизм – возможность объектов с одинаковой спецификацией иметь различную реализацию (использование одного имени для решения двух или более схожих, но технически разных задач).

Пример в Java:

package OOP.Polymorphism;  
  
public abstract class Animal {  
 public abstract void animalSound();  
}

package OOP.Polymorphism;  
  
public class Lion extends Animal {  
  
 @Override  
 public void animalSound() {  
 System.*out*.println("Roar");  
 }  
}

package OOP.Polymorphism;  
  
public class Dog extends Animal {  
  
 @Override  
 public void animalSound() {  
 System.*out*.println("Woof");  
 }  
}

package OOP.Polymorphism;  
  
import OOP.Inheritance.Child;  
  
public class Examples {  
  
 public static void main(String[] args){  
 //Создаем два объекта Animal у которых можно вызвать функцию animalSound()  
 Animal lion = new Lion();  
 Animal dog = new Dog();  
 lion.animalSound();  
 dog.animalSound();  
 }  
}

Вывод:

Roar

Woof

Как видно одна и таже функция имеет разный функционал