**Проблема:**

класс Shape означает фигуру (не конкретную).

Мы можем создать объект класса Shape. То есть фактически мы создаём в программном коде объект Фигура (не квадрат или треугольник, а просто "Фигура")

**Абстрактный класс** - класс объявленный с ключевым словом ***abstract***

Пример:

**abstract** class Shape{}

Абстрактный класс является таким же классом как и обычные, но с некоторыми отличиями.

Отличия:

1) Нельзя создавать объект абстрактного класса

**Shape s=new Shape();**

2) В абстрактном классе можно объявить абстрактный метод

Пример:

abstract public void draw(int x,int y);

У абстрактного метода **НЕТ** тела.

Примечания:

а) Абстрактный метод не может быть реализован в теле абстрактного класса

б) Абстрактный метод **ДОЛЖЕН** быть реализован в потомках класса или потомок тоже должен быть абстрактным классом

в) Абстрактные методы могут быть только в абстрактных классах.

Абстрактные классы часто применяются для реализации полиморфизма (для указания общего типа) и инкапсуляции некоторых действий в базовом классе.

**Интерфейсы**

Задача:

Создадим абстрактный класс Animal

Создадим класс Horse- потомок Animal с методом run()

Создадим класс Bird – потомок Animal с методом fly()

Нужно создать класс Pegasus который будет иметь методы run() и fly()

По сути необходимо реализовать множественное наследование: класс Pegasus одновременно является как классом Horse так и классом Bird (то есть он должен унаследовать признаки обоих).

В языке Java нет множественного наследования.

Решения:

1) унаследовать Pegasus от Horse и скопировать метод fly() из класса Bird

2) унаследовать Pegasus от Bird и скопировать метод run()из класса Horse

Проблема:

Класс ZooCoach – тренер в зоопарке. Тренер в зоопарке хочет использовать животных, чтобы они показывали, что они умеют.

Для этого в классе ZooCoach реализованы методы:

void whoCanFly(Bird b){

b.fly();

}

void whoCanRun(Horse h){

h.run();

}

**Суть проблемы: Пегас либо Птица либо Лошадь!**

**Решение: Интерфейсы**

Интерфейс объявляется при помощи служебного слова interface:

interface InterFaceName{

Сигнатуры методов **– БЕЗ ТЕЛА МЕТОДА**

}

Правила именования интерфейсов совпадают с правилами именования классов.

Интерфейсы содержат в себе описания (сигнатуры) методов.

Классы могут РЕАЛИЗОВЫВАТЬ интерфейс - то есть реализовывать ВСЕ методы указанные в теле интерфейса.

Класс может реализовывать сколько угодно интерфейсов.

Следовательно, если класс реализует интерфейс, то он имеет реализованные методы с ТОЙ ЖЕ сигнатурой что и указаны в интерфейсе.

Пример:

Runner – интерфейс с функцией run

interface Runner{

void run();

}

Flyer – интерфейс с функцией fly

interface Flyer{

void fly();

}

Класс Animal – базовый класс в иерархии

abstract class Animal{}

class Horse **extends Animal** ***implements Runner***{

public ***void run()***{

System.out.println("I can run");

}

}

class Bird **extends Animal** ***implements Flyer***{

public ***void fly()***{

System.out.println("I can fly");

}

}

class Pegasus **extends Animal** ***implements Flyer,Runner***{

public ***void fly()***{

System.out.println("I can fly like Pegasus");

}

public **void run()**{

System.out.println("I can run like Pegasus");

}

}

public class MagicZoo{

static void whoCanRun(***Runner runAble***){

runAble.***run***();

}

static void whoCanFly(***Flyer flyAble***){

flyAble.***fly***();

}

public static void main(String [] a){

Horse h=new Horse();

Bird b=new Bird();

Pegasus p=new Pegasus();

Plane plane=new Plane();

System.out.println("Horse can run!");

whoCanRun(h);

System.out.println("Bird can fly!");

whoCanFly(b);

System.out.println("Pegasus can fly and run!");

whoCanRun(p);

whoCanFly(p);

System.out.println("Plane also can fly");

whoCanFly(plane);

}

}

Можно указать интерфейс как тип параметра функции и мы ничего не знаем про класс, который реализует его кроме того что он реализует функции этого интерфейса.

Классы реализующие интерфейс **не обязаны** находится в одной иерархии наследования.

Что ещё:

Наследование

Инициализация и завершение, финализация, сборка мусора.

Исключения

Внутренние классы

Управление доступом (пакеты)

Параметризация (Generics)

Ввод/вывод в Java