Практика 11

1. Программу, должна давать имена всем котикам, выпускаемым на котофабрике.Для этого нужно:
   1. Считывать **строки** (**параметры**) с консоли, пока пользователь не введет пустую **строку** (**Enter**).
   2. Для каждого параметра (имени кота):
   создать объект cat класса Cat, который равен коту из getCatByKey(String **параметр**).
   вывести на экран cat.toString().

```
public class Solution {
  public static void main(String[] args) throws Exception {
    //напишите тут ваш код
  }
  static class CatFactory {
    static Cat getCatByKey(String key) {
      Cat cat;
      switch (key) {
        case "vaska":
          cat = new MaleCat("Василий");
          break;
        case "murka":
          cat = new FemaleCat("Мурочка");
          break;
        case "kiska":
          cat = new FemaleCat("Кисюлька");
          break;
        default:
          cat = new Cat(key);
          break;
      }
      return cat;
    }
  }
  static class Cat {
    private String name;
    protected Cat(String name) {
      this.name = name;
    }
    public String getName() {
      return this.name;
    }
    public String toString() {
      return "Я уличный кот " + getName();
    }
  }
  static class MaleCat extends Cat {
    MaleCat(String name) {
      super(name);
    }
    public String toString() {
      return "Я - солидный кошак по имени " + getName();
    }
  }
  static class FemaleCat extends Cat {
    FemaleCat(String name) {
      super(name);
    }
    public String toString() {
      return "Я - милая кошечка по имени " + getName();
    }
  }
}
```

2. Программа, которая помогает выбрать, что съесть на обед.
   Для этого нужно:
   1. Реализовать интерфейс Selectable в классе **Food**.
   2. Метод onSelect() должен выводить на экран фразу "**The food was selected**".
   3. Подумайте, какие методы можно вызвать для переменной food, а какие для — selectable.
   4. В методе foodMethods вызовите методы onSelect, onEat, если это возможно.
   5. В методе selectableMethods вызовите методы onSelect, onEat, если это возможно.
   6. Не используйте явное приведение типов.

```
public class Solution {
   public static void main(String[] args) {
      Food food = new Food();
      Selectable selectable = new Food();
      Food newFood = (Food) selectable;

      foodMethods(food);
      selectableMethods(selectable);
   }

   public static void foodMethods(Food food) {
      //тут добавьте вызов методов для переменной food
   }

   public static void selectableMethods(Selectable selectable) {
      //тут добавьте вызов методов для переменной selectable
   }

   interface Selectable {
      void onSelect();
   }

   static class Food {
      public void onEat() {
         System.out.println("The food was eaten");
      }
   }
}
```

3. В этой задаче нужно:
   1. Создать интерфейс Bridge с методом int getCarsCount().
   2. Создать классы **WaterBridge** и **SuspensionBridge**, которые реализуют интерфейс Bridge.
   3. Метод getCarsCount() должен возвращать любое фиксированное значение типа int.
   4. Метод getCarsCount() должен возвращать различные значения для различных классов.
   5. В классе **Solution** создать публичный метод println(Bridge **bridge**).
   6. В методе println() вывести на консоль значение getCarsCount() для объекта bridge.

```
public class Solution {
   public static void main(String[] args) {
      println(new WaterBridge());
      println(new SuspensionBridge());
   }

   //add println method here
}
```

4. Программа, которая поможет определить, какое вино пить по какому случаю. Для этого нужно:
   1. Создать абстрактный класс **Drink** с реализованным методом `public void taste()`, который выводит в консоль "**Вкусно**".
   2. Создать класс **Wine**, наследуемый от **Drink**, с реализованным методом `public String getHolidayName()`, который возвращает строку "**День Рождения**".
   3. Создать класс **SparklingWine**, наследуемый от **Wine**, с реализованным методом `public String getHolidayName()`, который возвращает строку "**Новый Год**".
   4. Написать реализацию методов `getDeliciousDrink`, `getWine`, `getSparklingWine`.

```
public class Solution {
   public static void main(String[] args) {
      getDeliciousDrink().taste();
      System.out.println(getWine().getHolidayName());
      System.out.println(getSparklingWine().getHolidayName());
      System.out.println(getWine().getHolidayName());
   }

   public static Drink getDeliciousDrink() {

   }

   public static Wine getWine() {

   }

   public static Wine getSparklingWine() {

   }
}
```

5. Программа, которая определит, чем заняться тому или иному человеку. Для этого нужно:
   1. Ввести **[в цикле]** с клавиатуры несколько строк (**ключей**). Строки (ключи) могут быть такими: "**user**", "**loser**", "**coder**", "**proger**". Ввод окончен, когда строка не совпадает ни с одной из выше указанных.
   2. Для каждой **введенной** строки нужно:
   а. Создать соответствующий объект [см. **Person.java**], например, для строки "**user**" нужно создать объект класса **User**.
   б. Передать этот объект в метод doWork.
   3. Написать реализацию метода doWork, который:
   а. Вызывает метод live() у переданного объекта, если этот объект (person) имеет тип **User**.
   б. Вызывает метод doNothing(), если person имеет тип **Loser**.
   в. Вызывает метод writeCode(), если person имеет тип **Coder**.
   г. Вызывает метод enjoy(), если person имеет тип **Proger**.

см. Следущий слайд

```java
public class Solution {
   public static void main(String[] args) throws Exception {
      BufferedReader reader = new BufferedReader(new InputStreamReader(System.in));
      Person person = null;
      String key = null;
      //тут цикл по чтению ключей, пункт 1
      {
         //создаем объект, пункт 2
         doWork(person); //вызываем doWork
      }
   }
   public static void doWork(Person person) {
      // пункт 3
   }
}
```

```java
public interface Person {
   class User implements Person {
      void live() {
         System.out.println("I usually just live.");
      }
   }

   class Loser implements Person {
      void doNothing() {
         System.out.println("I usually do nothing.");
      }
   }

   class Coder implements Person {
      void writeCode() {
         System.out.println("I usually write code.");
      }
   }

   class Proger implements Person {
      void enjoy() {
         System.out.println("It's a wonderful life!");
      }
   }

}
```

6. Напишите реализацию метода printMainInfo, чтобы:
   1. Если в метод передают объект типа **Drawable**, у этого объекта вызывался метод draw.
   2. Если в метод передают объект типа **Movable**, у этого объекта вызывался метод move.

```
public class Solution {
   public static void main(String[] args) {
      Object obj = new Circle();
      Movable movable = (Movable) obj;
      Drawable drawable = new Rectangle();
      printMainInfo(drawable);
      printMainInfo(movable);
   }
   public static void printMainInfo(Object object) {
      //напишите тут ваш код
   }
   static interface Movable {
      void move();
   }
   static class Circle implements Movable {
      public void draw() {
         System.out.println("Can be drawn");
      }
      public void move() {
         System.out.println("Can be moved");
      }
   }
   static interface Drawable {
      void draw();
   }
   static class Rectangle implements Drawable {
      public void draw() {
         System.out.println("Can be drawn");
      }
      public void move() {
         System.out.println("Can be moved");
      }
   }
}
```

7. Соберем компьютер. Вот что нужно сделать:
   1. Создайте интерфейс CompItem.
   2. Добавьте в него метод String getName().
   3. Создайте классы **Keyboard**, **Mouse**, **Monitor**, которые реализуют интерфейс CompItem.
   4. Метод getName() должен возвращать имя класса, например, для класса **Keyboard** будет "**Keyboard**".
   5. Создайте класс **Computer**.
   6. В класс **Computer** добавьте приватное поле keyboard типа Keyboard.
   7. В класс **Computer** добавьте приватное поле mouse типа Mouse.
   8. В класс **Computer** добавьте приватное поле monitor типа Monitor.
   9. Создайте конструктор с тремя параметрами в классе **Computer** используя комбинацию клавиш Alt+Insert (для Windows) внутри класса (команда Constructor).
   10. Внутри конструктора инициализируйте все три поля (переменных) класса в соответствии с переданными параметрами.
   11. Создайте геттеры для полей класса **Computer** (в классе используй комбинацию клавиш Alt+Insert (для Windows) и выбери команду Getter).

```
public class Solution {
   public static void main(String[] args) {
      Computer computer = new Computer(new Keyboard(), new Mouse(), new Monitor());
      if (isWork(computer.getKeyboard()) &&
            isWork(computer.getMonitor()) &&
            isWork(computer.getMouse())) {
         System.out.println("Work!");
      }
   }
   public static boolean isWork(CompItem item) {
      System.out.println(item.getName());
      return item.getName() != null && item.getName().length() > 4;
   }
}
```

8. У нас есть кинофабрика, но она работает не в полную силу. Расширим ее функционал по аналогии с тем, что уже есть, и добавим чтение с консоли.
   1. Разобраться, что программа умеет делать.
   2. Все классы должны быть внутри класса **Solution**.
   3. Добавить классы **Cartoon**, **Thriller**.
   4. Разобраться, как мы получаем объект класса **SoapOpera** по ключу "**soapOpera**".
   5. Аналогично получению объекта SoapOpera сделать:
   а. добавить в MovieFactory.getMovie получение объекта Cartoon для ключа "**cartoon**".
   б. добавить в MovieFactory.getMovie получение объекта Thriller для ключа "**thriller**".
   6. Считать с консоли несколько ключей (строк). Важно: ввод заканчивается, как только вводится строка не совпадающая с одной из: "**cartoon**", "**thriller**", "**soapOpera**".
   7. Создать переменную movie типа **Movie** и для каждой введенной строки (**ключа**):
   а. получить объект используя MovieFactory.getMovie и присвоить его переменной movie.
   б.вывести на экран movie.getClass().getSimpleName().

см. Следущий слайд

```
public class Solution {
   public static void main(String[] args) throws Exception {
      //ввести с консоли несколько ключей (строк), пункт 7

      /*
8 Создать переменную movie класса Movie и для каждой введенной строки(ключа):
8.1 получить объект используя MovieFactory.getMovie и присвоить его переменной movie
8.2 вывести на экран movie.getClass().getSimpleName()
      */

   }

   static class MovieFactory {

      static Movie getMovie(String key) {
         Movie movie = null;

         //создание объекта SoapOpera (мыльная опера) для ключа "soapOpera"
         if ("soapOpera".equals(key)) {
            movie = new SoapOpera();
         }

         //напишите тут ваш код, пункты 5,6

         return movie;
      }
   }

   static abstract class Movie {
   }

   static class SoapOpera extends Movie {
   }

   //Напишите тут ваши классы, пункт 3
}
```

9. В этой задаче нужно:
   1. Реализовать метод cleanAllApartments().
   2. Пройтись по списку объектов **apartments**:
   для однокомнатных квартир (**OneRoomApt**) вызвать метод clean1Room();
   для двухкомнатных квартир (**TwoRoomApt**) вызвать метод clean2Rooms();
   для трехкомнатных квартир (**ThreeRoomApt**) вызвать метод clean3Rooms().

```
public class Solution {
   public static void main(String[] args) {
      List<Apartment> apartments = new ArrayList<Apartment>();
      apartments.add(new OneRoomApt());
      apartments.add(new TwoRoomApt());
      apartments.add(new ThreeRoomApt());

      cleanAllApartments(apartments);
   }

   public static void cleanAllApartments(List<Apartment> apartments) {
      //написать тут вашу реализацию пунктов 1-4
   }

   static interface Apartment {
   }

   static class OneRoomApt implements Apartment {
      void clean1Room() {
         System.out.println("1 room is cleaned");
      }
   }

   static class TwoRoomApt implements Apartment {
      void clean2Rooms() {
         System.out.println("2 rooms are cleaned");
      }
   }

   static class ThreeRoomApt implements Apartment {
      void clean3Rooms() {
         System.out.println("3 rooms are cleaned");
      }
   }
}
```

10. Напишем программу, которая определит, что умеют делать жители океана:
    1. Подумайте, как связаны интерфейсы Swimmable (**способен плавать**) и Walkable (**способен ходить**) с классом **OceanAnimal** (**животное океана**).
    2. Правильно расставьте наследование интерфейсов и класса **OceanAnimal**.
    3. Подумайте, как могут быть связаны классы **Orca** (**Косатка**), **Whale** (**Кит**), **Otter** (**Выдра**) с классом **OceanAnimal**.
    4. Расставьте правильно наследование между классами **Orca**, **Whale**, **Otter** и классом **OceanAnimal**.
    5. Подумайте, какой класс должен реализовать интерфейс Walkable и добавить интерфейс этому классу.
    6. Подумайте, какое животное еще не умеет плавать и добавить ему интерфейс Swimmable.

см. Следущий слайд

```java
public class Solution {
    public static void main(String[] args) {
        Swimmable animal = new Orca();
        animal.swim();
        animal = new Whale();
        animal.swim();
        animal = new Otter();
        animal.swim();
    }

    public static void test(Swimmable animal) {
        animal.swim();
    }

    interface Walkable {
        void walk();
    }

    interface Swimmable {
        void swim();
    }

    static abstract class OceanAnimal {
        public void swim() {
            OceanAnimal currentAnimal = (OceanAnimal) getCurrentAnimal();
            currentAnimal.displaySwim();
        }

        private void displaySwim() {
            System.out.println(getCurrentAnimal().getClass().getSimpleName() + " is swimming");
        }

        abstract Swimmable getCurrentAnimal();
    }

    static class Orca {
    }

    static class Whale {

    }

    static class Otter {

    }
}
```

1. Расширьте функциональность программы, которая позволит производить манипуляции с валютами.
   1. В абстрактном классе **Money** создайте приватное поле amount типа double.
   2. Создайте публичный геттер для поля amount(public double getAmount()), чтобы к этому полю можно было получить доступ извне класса **Money**.
   3. В отдельных файлах создайте классы **Hryvnia**, **Ruble** и **USD**, которые будут являться потомками класса **Money**.
   4. В классах **Hryvnia**, **Ruble** и **USD** реализуйте метод getCurrencyName() который будет возвращать название соответствующей валюты (**строку**) в виде аббревиатуры (**USD, UAH, RUB**).
   5. В классах **Hryvnia**, **Ruble** и **USD** реализуйте публичный (**public**) конструктор, который принимает один параметр и вызывает конструктор базового класса (**super**) с этим параметром.
   6. Заполните список **allMoney** объектами всех возможных в рамках условия задачи и функциональности программы валют.

```
public class Solution {
   public static void main(String[] args) {
      Person ivan = new Person("Иван");
      for (Money money : ivan.getAllMoney()) {
         System.out.println(ivan.name + " имеет заначку в размере " + money.getAmount() + " " + money.getCurrencyName());
      }
   }
   static class Person {
      public String name;
      Person(String name) {
         this.name = name;
         this.allMoney = new ArrayList<Money>();
         //напишите тут ваш код
      }
      private List<Money> allMoney;
      public List<Money> getAllMoney() {
         return allMoney;
      }
   }
}
```

12. У нас есть программа, которая должна заполнять список и выводить его определенным образом в консоли.
Сейчас она работает некорректно. Чтобы исправить программу:
1. Подумайте что делает метод main().
2. Создайте в классе **Solution** статические методы initList(**List<Number>** list), printListValues(**List<Number>** list), processCastedObjects(**List<Number>** list).
3. Найдите блок кода, который заполняет значениями список, и переместите его в метод initList.
4. Найдите блок кода, который в цикле for выводит на экран содержимое списка, и переместите его в метод printListValues.
5. Найдите блок кода, в котором для каждого объекта списка проверяется тип и выводятся сообщения на экран, и переместите его в метод processCastedObjects.
6. Исправьте **2** ошибки в методе printListValues так, чтобы на экран корректно выводилось содержимое переданного в качестве параметра списка.
7. Исправьте **2** ошибки в методе processCastedObjects, связанные с приведением типов:
для объекта типа Float нужно вывести "**Is float value defined?** " + **!([Float_object].isNaN())**.
для объекта типа Double нужно вывести "**Is double value infinite?** " + **[Double_object].isInfinite()**.
см. Следущий слайд

```java
public class Solution {
    public static void main(String[] args) {
        List<Number> list = new LinkedList<Number>();
        //3
        list.add(new Double(1000f));
        list.add(new Double("123e-445632"));
        list.add(new Float(-90 / -3));
        list.remove(new Double("123e-445632"));

        //4 - Исправь 2 ошибки
        for (int i = 0; i <= list.size(); i--) {
            System.out.println(list.get(i));
        }

        //5
        for (Number object : list) {
            //Исправь 2 ошибки
            if (object instanceof Float) {
                Double a = (Double) object;
                System.out.println("Is float value defined? " + !(a.isNaN()));
            } else if (object instanceof Double) {
                Float a = (Float) object;
                System.out.println("Is double value infinite? " + a.isInfinite());
            }
        }
    }
}
```

13. Заполните список **exceptions** десятью различными исключениями. Первое исключение уже реализовано в методе initExceptions.

```
public class Solution {
   public static List<Exception> exceptions = new ArrayList<Exception>();

   public static void main(String[] args) {
      initExceptions();

      for (Exception exception : exceptions) {
         System.out.println(exception);
      }
   }

   private static void initExceptions() {   //the first exception
      try {
         float i = 1 / 0;

      } catch (Exception e) {
         exceptions.add(e);
      }

      //напишите тут ваш код

   }
}
```

14. Найдем наибольший общий делитель (**НОД**). Для этого:
    1. Введите с клавиатуры **2** целых положительных числа.
    2. Выведите в консоли наибольший общий делитель.

15. Напишем Фабрику (**Factory**) по производству **кур** (**Hen**):ё
    1. Создайте класс **Hen**.
    а. Сделайте его абстрактным.
    б. Добавьте в класс абстрактный метод int getCountOfEggsPerMonth().
    в. Добавьте в класс метод String getDescription(), который возвращает строку "**Я - курица.**".
    2. Создайте класс **RussianHen**, который наследуется от **Hen**.
    3. Создайте класс **UkrainianHen**, который наследуется от **Hen**.
    4. Создайте класс **MoldovanHen**, который наследуется от **Hen**.
    5. Создайте класс **BelarusianHen**, который наследуется от **Hen**.
    6. В каждом из четырех последних классов напишите свою реализацию метода getCountOfEggsPerMonth.
    7. Методы должны возвращать **количество яиц в месяц** от данного типа куриц.
    8. В каждом из четырех последних классов напишите свою реализацию метода getDescription. **Методы должны возвращать строку вида: <getDescription() родительского класса> + <" Моя страна - Sssss. Я несу N яиц в месяц.">**
    где Sssss - название страны
    где N - количество яиц в месяц
    9. В классе **HenFactory** реализуйте метод **getHen**, который возвращает соответствующую стране породу кур.

см. Следущий слайд

```
public class Solution {
   public static void main(String[] args) {
      Hen hen = HenFactory.getHen(Country.BELARUS);
      hen.getCountOfEggsPerMonth();
   }

   static class HenFactory {

      static Hen getHen(String country) {
         Hen hen = null;
         //напишите тут ваш код
         return hen;
      }
   }

}



public interface Country {
   String UKRAINE = "Ukraine";
   String RUSSIA = "Russia";
   String MOLDOVA = "Moldova";
   String BELARUS = "Belarus";
}
```