Практика 10

1. Посмотрите на программу и исправьте ее, чтобы она компилировалась и работала:
   Вынесите реализацию метода initializeIdAndName в класс **User**.
   Сделайте так, чтобы initializeIdAndName в классе **User** возвращал тип User.

```
public class Solution {
   public static void main(String[] args) throws Exception {
      System.out.println(Matrix.NEO);
      System.out.println(Matrix.TRINITY);
   }

   static class Matrix {
      public static DBObject NEO = new User().initializeIdAndName(1, "Neo");
      public static DBObject TRINITY = new User().initializeIdAndName(2, "Trinity");
   }

   interface DBObject {
      DBObject initializeIdAndName(long id, String name) {
         this.id = id;
         this.name = name;
         return this;
      }
   }

   static class User implements DBObject {
      long id;
      String name;

      @Override
      public String toString() {
         return String.format("The user's name is %s, id = %d", name, id);
      }
   }

}
```

2. Исправьте **4** ошибки в программе, чтобы она компилировалась.

```
public class Solution {

   public static void main(String[] args) {

      System.out.println(new Dream().HOBBY.toString());
      System.out.println(new Hobby().toString());

   }

   interface Desire {
   }

   interface Dream {
      private static Hobby HOBBY = new Hobby();
   }

   class Hobby extends Desire implements Dream {
      static int INDEX = 1;

      @Override
      public String toString() {
         INDEX++;
         return "" + INDEX;
      }
   }

}
```

3. В классе **StringObject** реализуйте интерфейс SimpleObject с параметром типа String.

```
public class Solution {
   public static void main(String[] args) throws Exception {
   }

   interface SimpleObject<T> {
      SimpleObject<T> getInstance();
   }

   class StringObject //допишите здесь ваш код
   {
   }
}
```

4. В этой задаче нужно:
   1. Создать интерфейс Person.
   2. Добавить в него метод isAlive(), который проверяет, жив человек или нет.
   3. Подумать, какой тип должен возвращать этот метод.
   4. Создать интерфейс Presentable.
   5. Унаследовать интерфейс Presentable от интерфейса Person.

5. В этой задаче нужно:
   1. Создать интерфейс CanMove с методом speed.
   2. Сделать так, чтобы speed возвращал значение типа Double и ничего не принимал в качестве аргументов.
   3. Создать и унаследовать интерфейс CanFly от интерфейса CanMove.
   4. Добавить в интерфейс CanFly метод speed.
   5. Убедиться, что speed возвращает значение типа Double и принимает один параметр типа CanFly.

6. Добавьте интерфейсы Secretary и Boss к классам **Manager** и **Subordinate**. По одному на каждый. Подумайте, кому какой. Унаследуйте интерфейсы Secretary и Boss от интерфейсов Person и HasManagementPotential так, чтобы все методы у классов **Manager** и **Subordinate** были объявлены в каком-то интерфейсе.

```
public class Solution {
   public static void main(String[] args) throws Exception {
   }

   interface Person {
      void use(Person person);

      void startToWork();
   }

   interface HasManagementPotential {
      boolean inspiresOthersToWork();
   }

   interface Secretary {
   }

   interface Boss {
   }

   class Manager {
      public void use(Person person) {
         person.startToWork();
      }

      public void startToWork() {
      }

      public boolean inspiresOthersToWork() {
         return true;
      }
   }

   class Subordinate {
      public void use(Person person) {
      }

      public void startToWork() {
      }
   }
}
```

7. В этой задаче нужно:
   1. Создать класс **EnglishTranslator**, который наследуется от **Translator**.
   2. Реализовать все абстрактные методы.
   3. Подумай, что должен возвращать метод getLanguage.
   4. Сделать так, чтобы программа выводила: "**Я переводчик с английского**" путем вызова метода translate у объекта типа EnglishTranslator.

```
public class Solution {
    public static void main(String[] args) throws Exception {
        EnglishTranslator englishTranslator = new EnglishTranslator();
        System.out.println(englishTranslator.translate());
    }

    public static abstract class Translator {
        public abstract String getLanguage();

        public String translate() {
            return "Я переводчик с " + getLanguage();
        }
    }

}
```

8. Исправьте класс **BigFox** так, чтобы программа компилировалась.

```
public class Solution {

   public static void main(String[] args) throws Exception {
      Fox bigFox = new BigFox();
      System.out.println(bigFox.getName());
      System.out.println(bigFox.getColor());

   }

   public interface Animal {
      Color getColor();
   }

   public static abstract class Fox implements Animal {
      public String getName() {
         return "Fox";
      }
   }

   public static class BigFox {
      public Color() {
         return Color.GRAY;
      }
   }

}
```

9. Создайте классы **Dog**, **Cat** и **Mouse**. Реализуйте интерфейсы в добавленных классах, учитывая что:
   1. **Кот** (**Cat**) может передвигаться, может кого-то съесть и может быть сам съеден.
   2. **Мышь** (**Mouse**) может передвигаться, и ее могут съесть;
   3. **Собака** (**Dog**) может передвигаться и съесть кого-то.

```
public class Solution {
   public static void main(String[] args) {

   }

   //может двигаться
   public interface Movable {
      void move();
   }

   //может быть съеден
   public interface Edible {
      void beEaten();
   }

   //может кого-нибудь съесть
   public interface Eat {
      void eat();
   }
}
```

10. Удалите все некорректные строки в интерфейсе Button.
    В интерфейсе Button должно остаться корректное объявление метода onPress.

```
public class Solution {
   public static void main(String[] args) throws Exception {
      System.out.println(SimpleObject.NAME);
      System.out.println(Button.NAME);
   }

   interface SimpleObject {
      String NAME = "SimpleObject";

      void onPress();
   }

   interface Button extends SimpleObject {
      final String NAME = "Submit";

      public void onPress();

      protected void onPress();

      void onPress();

      private void onPress();

      protected String onPress(Object o);

      String onPress(Object o);

      private String onPress(Object o);

   }
}
```

11. Реализуйте в классе **Today** интерфейс Weather. Подумайте, как связан параметр **type** с методом getWeatherType(). Обратите внимание, что интерфейсы Weather и WeatherType реализованы в отдельных файлах.

```
public class Solution {
   public static void main(String[] args) {
      System.out.println(new Today(WeatherType.CLOUDY));
      System.out.println(new Today(WeatherType.FOGGY));
      System.out.println(new Today(WeatherType.FREEZING));
   }

   static class Today {
      private String type;

      Today(String type) {
         this.type = type;
      }

      @Override
      public String toString() {
         return String.format("Today it will be %s", this.getWeatherType());
      }
   }
}
```

Weather.java

```
public interface Weather {
   String getWeatherType();
}
```

WeatherType.java

```
public interface WeatherType {
   String CLOUDY = "Cloudy";
   String FOGGY = "Foggy";
   String FREEZING = "Freezing";
}
```

12. В этой задаче нужно:
    1. Считать с консоли **имя файла**.
    2. Вывести в консоли (на экран) содержимое файла.
    3. Освободить ресурсы. Закрыть **поток чтения** с файла и **поток ввода** с клавиатуры.

13. В этой задаче тебе нужно:
    1. Прочесть с консоли имя файла.
    2. Считывать строки с консоли, пока пользователь не введет строку "**exit**".
    3. Вывести абсолютно все введенные строки в файл, каждую строчку — с новой строки.

14. В этой задаче нужно:
    1. Реализовать интерфейс DBObject в классе **User**.
    2. Реализовать метод initializeIdAndName так, чтобы программа работала и выводила на экран: "**The user's name is Neo, id = 1**".
    3. Метод toString и метод main менять нельзя.
    4. Подумать, что должен возвращать метод initializeIdAndName в классе **User**.

    Обратите внимание: методы toString() и main() менять нельзя.

```
public class Solution {
    public static void main(String[] args) throws Exception {
        System.out.println(Matrix.NEO);
    }

    static class Matrix {
        public static DBObject NEO = new User().initializeIdAndName(1, "Neo");
    }

    interface DBObject {
        DBObject initializeIdAndName(long id, String name);
    }

    static class User {
        long id;
        String name;

        @Override
        public String toString() {
            return String.format("The user's name is %s, id = %d", name, id);
        }
    }
}
```

15. Переделайте наследование в классах и интерфейсах так, чтобы программа компилировалась и продолжала делать то же самое.Класс **Hobby** должен наследоваться от интерфейсов Desire, Dream.

```
public class Solution {
   public static void main(String[] args) {
      System.out.println(Dream.HOBBY.toString());
      System.out.println(new Hobby().INDEX);
   }

   interface Desire {
   }

   interface Dream implements Hobby {
      public static Hobby HOBBY = new Hobby();
   }

   static class Hobby extends Desire, Dream {
      static int INDEX = 1;

      @Override
      public String toString() {
         INDEX++;
         return "" + INDEX;
      }
   }
}
```

16. Создайте класс **StringObject** и реализуйте в нем интерфейс SimpleObject с параметром типа String. Программа должна компилироваться.

```
public class Solution {
   public static void main(String[] args) throws Exception {
      SimpleObject<String> stringObject = new StringObject();
   }

   interface SimpleObject<T> {
      SimpleObject<T> getInstance();
   }
}
```

17. Реализуйте интерфейс Updatable в классе **Screen**.

```
public class Solution {
   public static void main(String[] args) throws Exception {
   }

   interface Selectable {
      void onSelect();
   }

   interface Updatable extends Selectable {
      void refresh();
   }

   class Screen {

   }
}
```

18. Унаследуйте Fox от интерфейса Animal. Поменяйте код так, чтобы в классе Fox был только один метод - getName. Учтите, что методы удалять нельзя.

```
public class Solution {
   public static void main(String[] args) throws Exception {
   }

   public interface Animal {
      Color getColor();

      Integer getAge();
   }

   public static class Fox {
      public String getName() {
         return "Fox";
      }
   }
}
```

19. Исправьте классы Fox и BigFox так, чтобы программа компилировалась. В решении этой задачи не нужно создавать экземпляры базового класса. Метод main не изменяйте.

```
public class Solution {

   public static void main(String[] args) throws Exception {
      Fox bigFox = new BigFox();
      System.out.println(bigFox.getName());
      System.out.println(bigFox.getColor());
   }

   public interface Animal {
      Color getColor();
   }

   public static class Fox implements Animal {
      public String getName() {
         return "Fox";
      }
   }

   public abstract static class BigFox {

   }

}
```

20. В этой задаче нужно:
    1. Ввести имя файла с консоли.
    2. Прочитать из него набор чисел.
    3. Вывести в консоли только четные, отсортированные по возрастанию.

    Пример ввода:

    5
    8
    -2
    11
    3
    -5
    2
    10

    Пример вывода:

    -2
    2
    8
    10

21. В этой задаче:
    1. Реализуйте интерфейс RepkaItem в классе **Person**.
    2. В классе **Person** реализуйте метод pull(Person person), который выводит фразу типа '**<name> за <person>**'.**Пример:**
    Бабка за Дедку
    Дедка за Репку
    3. Исправьте логическую ошибку цикла в методе tell класса **RepkaStory**.
    4. Выполните метод main.

```
public class Solution {
   public static void main(String[] args) {
      List<Person> plot = new ArrayList<Person>();
      plot.add(new Person("Репка", "Репку"));
      plot.add(new Person("Дедка", "Дедку"));
      plot.add(new Person("Бабка", "Бабку"));
      plot.add(new Person("Внучка", "Внучку"));
      RepkaStory.tell(plot);
   }
}
```

RepkaItem.java

```
public interface RepkaItem {
   public String getNamePadezh();
}
```

см. продолжение.

Person.java

```
public class Person {
   private String name;
   private String namePadezh;

   public Person(String name, String namePadezh) {
      this.name = name;
      this.namePadezh = namePadezh;
   }

   public String getName() {
      return name;
   }

   public void setName(String name) {
      this.name = name;
   }

   public String getNamePadezh() {
      return namePadezh;
   }

   public void setNamePadezh(String namePadezh) {
      this.namePadezh = namePadezh;
   }
}
```

RepkaStory.java

```
public class RepkaStory {
   static void tell(List<Person> items) {
      Person first;
      Person second;
      for (int i = items.size() - 1; i > 0; i--) {
         first = items.get(i - 1);
         second = items.get(i);
         first.pull(second);
      }
   }
}
```