Лабораторная работа 3. Работа с классами и их документирование

1. Основные теоретические сведения

1.1. Классы в Java

1.1.1. Объявление классов

Класс определяется с помощью ключевого слова class:

```
class Имя_класса{
        //Поля
        //Методы
}
```

Для хранения состояния объекта в классе применяются *поля* или *переменные* класса. Для определения поведения объекта в классе применяются *методы*.

Кроме обычных методов классы могут определять специальные методы, которые называются *конструкторами*. Конструкторы вызываются при создании нового объекта данного класса. Конструкторы выполняют инициализацию объекта:

```
class Имя_класса{
        //Конструктор
        Имя_класса(параметры){//...}
}
```

Если в классе не определено ни одного конструктора, то для этого класса автоматически создается конструктор без параметров.

Java не допускает использование деструкторов. Но благодаря наличию подсистемы сборки мусора потребность в функциях деструктора очень незначительна.

Для создания объекта используется оператор **new**:

Имя_класса Идентификатор = **new** Имя_класса().

Ключевое слово **this** представляет ссылку на текущий экземпляр класса. Через это ключевое слово мы можем обращаться к переменным, методам объекта, а также вызывать его конструкторы.

Кроме конструктора начальную инициализацию объекта вполне можно было проводить с помощью инициализатора объекта. Инициализатор выполняется до любого конструктора:

```
class Имя_класса {
      /*начало блока инициализатора*/
    {
      //поле= значение;
      //...
    }
    /*конец блока инициализатора*/

   // Крнструкторы
   //...
}
```

1.1.2. Модификаторы доступа и инкапсуляция

Java предоставляет ряд модификаторов доступа, чтобы задать уровни доступа для классов, переменных, методов и конструкторов. Существует четыре уровня доступа:

1. Видимый в пакете (стоит по умолчанию и модификатор не требуются).
2. Видимый только для класса (**private**).
3. Видимый для всех (**public**).
4. Видимый для пакета и всех подклассов (**protected**).

Модификатор private является наиболее ограничивающим уровнем доступа. Класс и интерфейсы не могут быть **private**. Переменные, объявленные как **private**, могут быть доступны вне класса, если получающие их открытые (**public**) методы присутствуют в классе. Использование модификатора **private** в Java является основным способом, чтобы скрыть данные.

Модификатор public — класс, метод, конструктор, интерфейс и т.д. объявленные как **public** могут быть доступны из любого другого класса. Поэтому поля, методы, блоки, объявленные внутри **public** класса могут быть доступны из любого класса, принадлежащего к "вселенной" Java. Тем не менее, если к **public** классу в другом пакете мы пытаемся получить доступ, то **public** класс приходится импортировать. Благодаря наследованию классов, в Java все публичные (**public**) методы и переменные класса наследуются его подклассами.

Модификатор protected — переменные, методы и конструкторы, которые объявляются как **protected** в суперклассе, могут быть доступны только для подклассов в другом пакете или для любого класса в пакете класса **protected**. Модификатор доступа **protected** в Java не может быть применен к классу и интерфейсам. Методы и поля могут быть объявлены как protected, однако методы и поля в интерфейсе не могут быть объявлены как **protected**. Доступ **protected** дает подклассу возможность использовать вспомогательный метод или переменную, предотвращая неродственный класс от попыток использовать их.

Если переменная имеет уровень доступа **private**, к ней невозможно обратиться извне класса, в котором она объявлена. Но все равно необходим способ обращения к **private** переменным из другого класса, иначе такие изолированные переменные не будут иметь смысла. Это достигается с помощью объявления специальных **public** методов. Методы, которые возвращают значение переменных, называются *геттеры*. Методы, которые изменяют значение свойств, называются *сеттеры*.

Существуют правила объявления таких методов, рассмотрим их:

- Если свойство НЕ типа boolean, префикс геттера должно быть get. Например: **getName**() это корректное имя геттера для переменной name.

- Если свойство типа boolean, префикс имени геттера может быть **get** или **is**. Например, **getPrinted**() или **isPrinted**() оба являются корректными именами для переменных типа boolean.
- Имя сеттера должно начинаться с префикса **set**. Например, **setName**() корректное имя для переменной name.
- Для создания имени геттера или сеттера, первая буква свойства должна быть изменена на большую и прибавлена к соответствующему префиксу (**set**, **get** или **is**).
- Сеттер должен быть **public**, возвращать **void** тип и иметь параметр соответствующий типу переменной. Например: **public void setAge** (**int** age) { **this**.age = age;}
- Геттер метод должен быть **public**, не иметь параметров метода, и возвращать значение соответствующее типу свойства. Например: **public int getAge**() { return age;}

В языке Java при проектировании классов принято ограничивать уровень доступа к переменным с помощью модификаторов **private** или **protected** и обращаться к ним через геттеры и сеттеры.

1.1.3. Наследование классов

Наследование (inheritance) — механизм, который позволяет описать новый класс на основе существующего (родительского). При этом свойства и функциональность родительского класса заимствуются новым классом.

Чтобы объявить один класс наследником от другого, надо использовать после имени класса-наследника ключевое слово **extends**, после которого идет имя базового класса:

class *Child* **extends** *Parent* {}

В Java нет множественного наследования от классов, но есть множественное наследование интерфейсов.

В Java есть особенность, связанная с наследованием конструкторов и их вызовом. В конструкторе дочернего класса, перед тем как будут выполнены выражения тела, неявно вызывается дефолтный (без

параметров) конструктор предка. Причем конструктор без параметров обязательно должен быть в родительском классе, если в дочернем предполагаются только конструкторы с аргументами, а родительские конструкторы не вызываются явно. В этом случае, если в родительском классе будут только конструкторы с аргументами, то в дочернем будет возникать ошибка.

Когда подклассу требуется сослаться на его непосредственный суперкласс используется ключевое слово **super**. У ключевого слова **super** имеются две общие формы:

- Для вызова конструктора суперкласса: **super**(*списокАргументов*);
- Для обращения к члену суперкласса, скрываемому членом подкласса: **super**.метод(*списокАргументов*)

Производный класс имеет доступ ко всем методам и полям базового класса (даже если базовый класс находится в другом пакете) кроме тех, которые определены с модификатором **private**. При этом производный класс также может добавлять свои поля и методы. При этом в Java наследовать только нестатические методы.

Дочерний класс может переопределить методы экземпляра своего родительского класса. Это называется ***переопределением*** метода. Переопределение метода выполняется для достижения полиморфизма во время выполнения программы.

1.1.4. Абстрактные классы

В абстрактном классе также можно определить поля и методы, в то же время нельзя создать объект или экземпляр абстрактного класса. Абстрактные классы призваны предоставлять базовый функционал для классов-наследников. А производные классы уже реализуют этот функционал.

При определении абстрактных классов используется ключевое слово **abstract**.

Кроме обычных методов абстрактный класс может содержать абстрактные методы. Такие методы определяются с помощью ключевого слова **abstract** и не имеют никакого функционала.

Производный класс обязан переопределить и реализовать все абстрактные методы, которые имеются в базовом абстрактном классе. Также следует учитывать, что если класс имеет хотя бы один абстрактный метод, то данный класс должен быть определен как абстрактный.

1.1.5. Интерфейсы

Интерфейсы определяют некоторый функционал, не имеющий конкретной реализации, который затем реализуют классы, применяющие эти интерфейсы. И один класс может применить множество интерфейсов.

Чтобы определить интерфейс, используется ключевое слово **interface**. Например:

```
interface Printable{
   void print();
}
```

Интерфейс может определять константы и методы, которые могут иметь, а могут и не иметь реализации. Методы без реализации похожи на абстрактные методы абстрактных классов.

Все методы интерфейса не имеют модификаторов доступа, но фактически по умолчанию доступ **public**, так как цель интерфейса - определение функционала для реализации его классом. Поэтому весь функционал должен быть открыт для реализации.

Чтобы класс применил интерфейс, надо использовать ключевое слово **implements**:

```
доступ class имя_класса [extends суперкласс]
                [implements интерфейс [,интерфейс ...]]{
//тело класса
}
```

В JDK 8 была добавлена такая функциональность как методы по умолчанию. И теперь интерфейсы кроме определения методов могут иметь их реализацию по умолчанию, которая используется, если класс,

реализующий данный интерфейс, не реализует метод. Например, создадим метод по умолчанию в интерфейсе Printable:

```
interface Printable {
    default void print(){
        System.out.println("Undefined printable");
    }
}
```

Метод по умолчанию - это обычный метод без модификаторов, который помечается ключевым словом **default**.

Начиная с JDK 8 в интерфейсах доступны статические методы - они аналогичны методам класса:

```
interface Printable {
    void print();
    static void read(){
        System.out.println("Read printable");
    }
}
```

Чтобы обратиться к статическому методу интерфейса также, как и в случае с классами, пишут название интерфейса и метод:

```
Printable.read();
```

Кроме методов в интерфейсах могут быть определены статические константы:

```
interface Stateable{
    int OPEN = 1;
    int CLOSED = 0;
    void printState(int n);
}
```

Хотя такие константы также не имеют модификаторов, но по умолчанию они имеют модификатор доступа **public static final**, и поэтому их значение доступно из любого места программы.

1.2. Построение диаграммы классов в среде IntelliJ IDEA

1.2.1. Установка плагина

Средства для построения диаграмм классов в IntelliJ IDEA версии Community отсутствует (хотя присутствуют в версии Ultimate). Устранить эту проблему позволяет установка плагина **simpleUMLCE**. Для его установки необходимо выполнить следующие действия:

1. Запустите IntelliJ IDEA.
2. Выполните команду **File-Settings… (Ctrl+Alt+S)**.
3. В окне "**Settings**" перейдите на вкладку **Plugins** и воспользуйтесь полем поиска для поиска плагина по названию.
4. Установите найденный плагин, воспользовавшись кнопкой **Install**.

1.2.2. Построение диаграммы

После установки плагина **simpleUMLCE** и перезапуска IntelliJ IDEA можно переходить к построению диаграмм. Для этого:

1. В дереве проекта выберите папку с классами.
2. В контекстном меню папки выберите команду **Add to simpleUML Diagram-New Diagram…**
3. В окне "**Choose a Name and Location**" задайте имя и расположение диаграммы.

1.3. Документирование кода с помощью JavaDoc

1.3.1. Комментарии Javadoc

Javadoc — стандартный генератор документации в HTML-формате из комментариев исходного кода.

Для создания описания к элементу (поле, класс, метод) используются специальный комментарий, расположенный выше этого элемента:

```
/**
 *
 */
```

В IntelliJ IDEA, если ввести /** и нажать **return**, она автоматически заполнит оставшуюся часть синтаксиса.

Документирование осуществляется с помощью специальных тегов (аннотаций).

Формат для добавления различных элементов выглядит следующим образом:

```
/**
* [short description]
* <p>
* [long description]
*
* [tags]
* [see also]
*/
```

1.3.2. Основные теги

Ниже приведены наиболее распространенные тэги, используемые в Javadoc.

@**author** Человек, который внес значительный вклад в код. Применяется только на уровне класса, пакета или обзора. Не включен в вывод Javadoc. Не рекомендуется включать этот тэг, поскольку авторство часто меняется.

@**param** Параметр, который принимает метод или конструктор.

@**deprecated** этим тэгом помечаются класс или метод, которые больше не используются. Такой тэг будет размещен на видном месте в Javadoc. Сопровождается тэгом @see или {@link}.

@**return** Что возвращает метод.

@**see** Создает список “см.также”. Используется в паре с тэгом {@link} для связи с содержимым.

{@**link**} Используется для создания ссылок на другие классы или методы. Пример: {@link Foo# bar} ссылается на метод bar, который принадлежит классу Foo. Для ссылки на метод в том же классе, просто добавляется #bar.

@**since** 2.0 Версия с момента добавления функции.

@**throws** Вид исключения, которое выдает метод. Обратим внимание, что для проверки этого тэга в коде должно быть указано исключение. В противном случае Javadoc выдаст ошибку. Тэг @exception является альтернативным тэгом.

@**Override** Используется с интерфейсами и абстрактными классами. Выполняет проверку, чтобы увидеть, является ли метод переопределением.

1.3.3. Порядок тэгов

Oracle предлагает следующий порядок тэгов:

@author (classes and interfaces)
@version (classes and interfaces)
@param (methods and constructors)
@return (methods)
@throws (@exception is an older synonym)
@see
@since
@serial
@deprecated

1.3.4. Пример

```
/** Класс служит для хранения объектов со свойствами
 * <b>maker</b> и <b>price</b>.
 * @autor Filippov Yakov
```

```
 * @version 1.0

*/
class Product{
	/** Свойство - производитель */
	private String maker;

	/** Свойство - цена */
	public double price;

	/** Создает новый пустой объект
	 * @see Product#Product(String, double)
	*/
	Product(){
		setMaker("");
		price=0;
	}

	/** Создает новый объект с заданными значениями
	 * @param maker - производитель
	 * @param price - цена
	 * @see Product#Product()
	*/

	Product(String maker,double price){
		this.setMaker(maker);
		this.price=price;
	}

		/** Функция для получения значения поля {@link
Product#maker}
	 * @return Возвращает название производителя
	 */

	public String getMaker() {
```

```
        return maker;
    }

    public void setMaker(String maker) {
        this.maker = maker;
    }

}
```

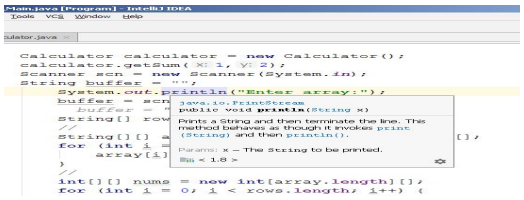
1.4. Настройка IntelliJ IDEA для работы с Javadoc

Для отображения подсказок Javadoc достаточно в редакторе кода щелкнуть по имени класса (метода) и нажать **Ctrl+Q**.

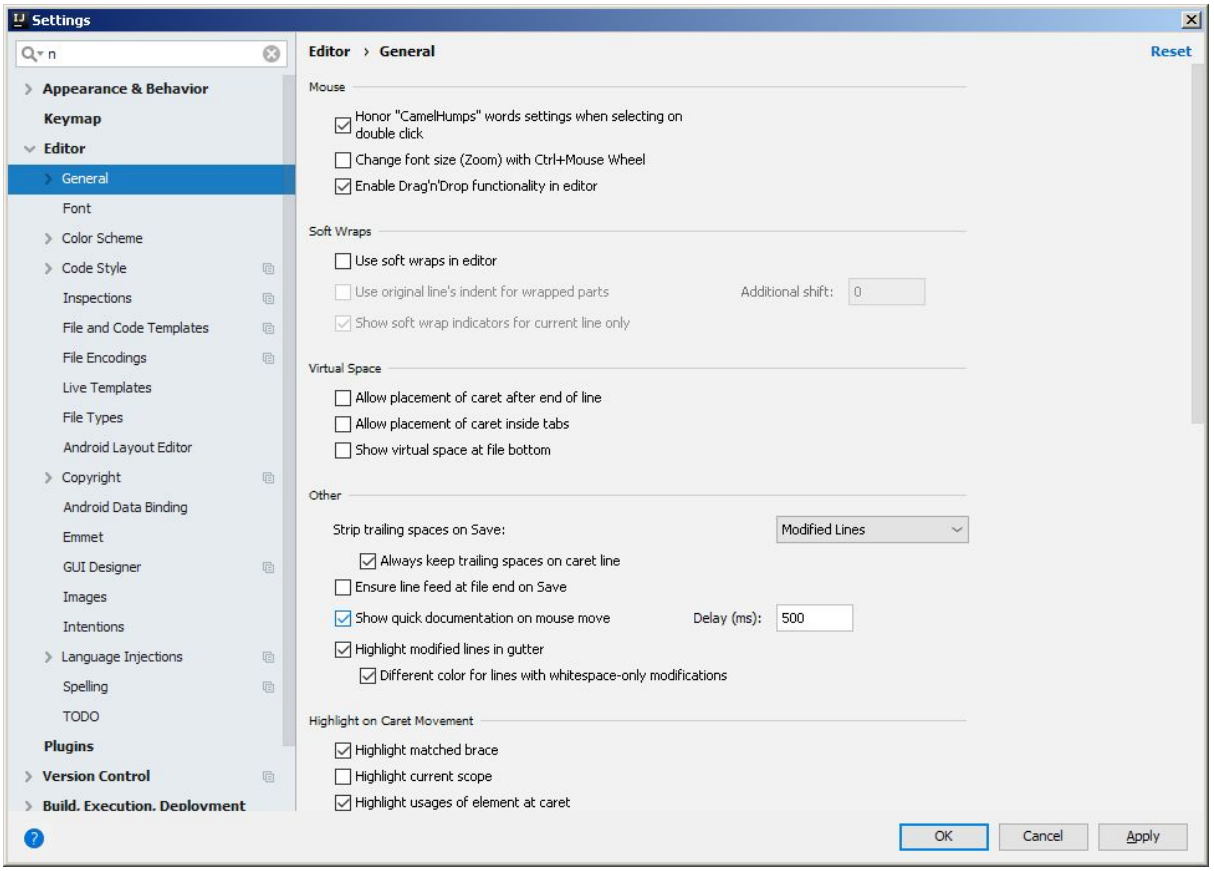
По умолчанию подсказка отображается в всплывающем окне, но такое поведение можно изменить и отображать ее в отдельной панели. Для этого можно воспользоваться кнопкой в виде шестеренки.

IntelliJ IDEA может быть настроена так, чтобы подсказка возникала при перемещении мыши над именем класса (метода). Для этого:

1. Выполните команду **File-Settings…(Ctrl+Alt+S)**.



2. В окне “**Settings**” перейдите на вкладку **Editor-General** и установите флажок **Show quick documentation on mouse move**:

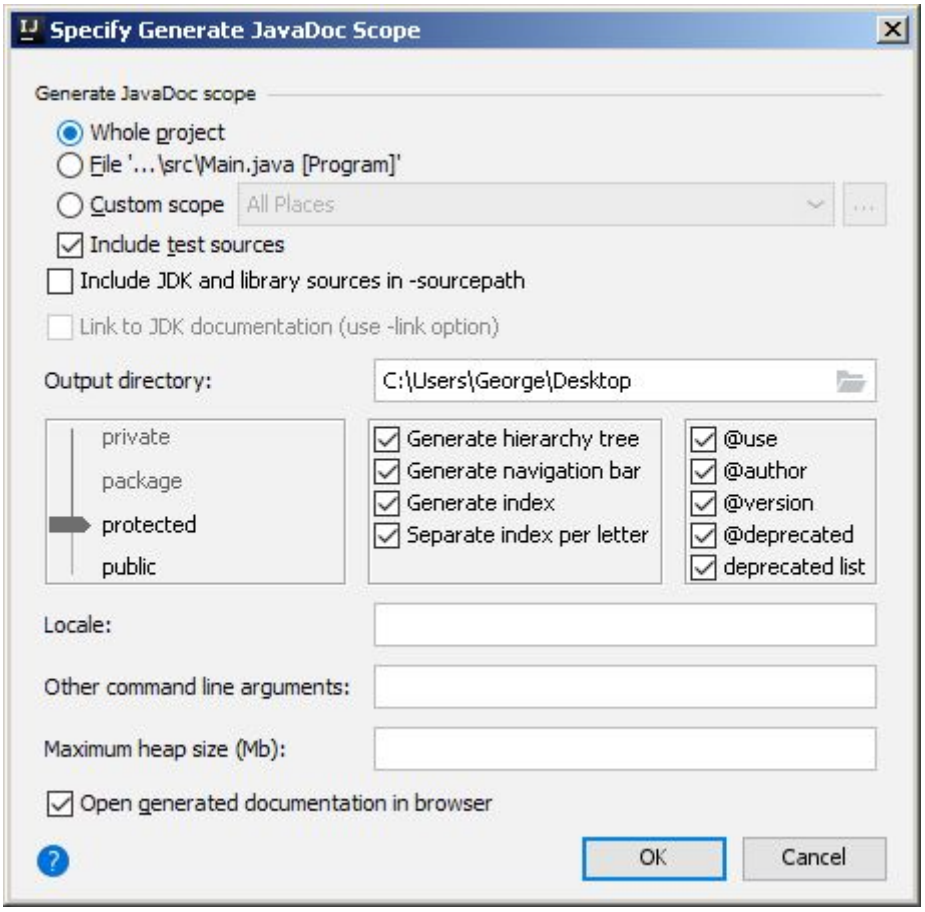


1.5. Генерация справки Javadoc

Для генерации файла справки в формате HTML необходимо:

1. Выполните команду **Tools-Generate JavaDoc**.
2. В окне “**Spesify Generate JavaDoc Scope**” установите необходимые параметры и выберите папку для сохранения файлов справки.
3. Нажмите **ОК** для запуска генерации справки.

По завершении генерации главная страница справки (index.html) открывается в браузере.



1.6. Горячие клавиши IntelliJ IDEA

Alt + Insert Генерация кода (Getters, Setters, Constructors, hashCode/equals, toString)

Ctrl + O Переопределение метода

Ctrl + I Реализация методов

Ctrl + Alt + T Поместить фрагмент кода в (if..else, try..catch, for, synchronized, etc.)

Ctrl + / Однострочное комментирование / раскомментирование

Ctrl + Shift + / Многострочное комментирование / раскомментирование

Ctrl + Alt + L Форматирование кода

Ctrl + Alt + O Удалить неиспользуемые импорты

2. Варианты заданий

№	Задачи	№	Задачи	№	Задачи	№	Задачи
1	1,5,16,40	**7**	8,13,18,22	13	2,14,28,38	19	8,15,22,34
2	2,13,17,39	**8**	9,14,23,35	14	3,5,29,39	20	9,13,21,35
3	3,15,18,38	**9**	5,10,24,34	15	4,15,26,30	21	10,14,20,36
4	4,14,19,37	**10**	11,15,25,35	16	5,13,25,31	22	5,11,19,37
5	5,6,20,36	**11**	12,14,26,36	17	5,7,24,32	23	12,15,18,38
6	7,15,17,21	**12**	1,13,27,37	18	14,6,23,33	24	1,13,17,33

1. Построить программу для работы с классом для хранения данных о кривой второго порядка - гиперболе. Программа должна обеспечивать: расчет y по x и наоборот, ввод значений, вывод значений.
2. Построить программу для работы с классом для хранения данных о кривой второго порядка - эллипсе. Программа должна обеспечивать простейшие функции: расчет y по x и наоборот, ввод значений, вывод значений.
3. Построить программу для работы с классом для хранения данных о кривой второго порядка - параболе. Программа должна обеспечивать простейшие функции: расчет y по x и наоборот, ввод значений, вывод значений.
4. Построить программу для работы с классом для хранения данных о погоде (направление, скорость ветра, температура, облачность, осадки). Программа должна обеспечивать простейшие функции: ввод значений, вывод значений.
5. Построить программу для работы с классом для хранения данных о сообщении на форуме (автор, тема, текст, время, дата создания и редактирование). Программа должна обеспечивать простейшие функции: ввод значений, редактирование, вывод значений.

6. Построить программу для работы с классом *Дата*. Программа должна обеспечивать простейшие функции: увеличение/уменьшение на 1 день, ввод значений, вывод значений.
7. Построить программу для работы с классом *Время*. Программа должна обеспечивать простейшие функции: увеличение/уменьшение на 1 час, минуту, секунду, ввод значений, вывод значений.
8. Построить программу для работы с классом *Правильная дробь*, который должен включать соответствующие поля: числитель, знаменатель. Программа должна обеспечивать простейшие функции: сложение, вычитание, умножение, деление, вывод дроби в удобной форме.
9. Построить программу для работы с классом *Комплексное число*. Класс должен включать соответствующие поля: вещественную и мнимую часть числа. Программа должна обеспечивать простейшие функции: сложение, вычитание, умножение, деление, вывод числа в удобной форме.
10. Создать класс типа *Прямоугольник*. Поля - высота и ширина. Функции-члены вычисляют площадь, периметр, устанавливает поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров.
11. Создать класс *Игра в крестики-нолики*. Поле класса - массив из (3x3) или целые числа. Ставить можно только на свободные.
12. Создать класс *Круг*. Поле - радиус. Функции-члены вычисляют площадь, длину окружности, устанавливает поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров.
13. Создать класс *Квадрат*. Поля - сторона. Функции-члены вычисляют площадь, периметр, устанавливает поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров.

14. Создать класс *Треугольник*. Поля - стороны. Функции-члены вычисляют площадь, периметр, высоты, устанавливает поля и возвращают значения. Функции-члены установки полей класса должны проверять корректность задаваемых параметров.
15. Создать класс *Линия на экране*, который имеет ширину и высоту. Поля – координаты начала и конца. Функции-члены вычисляют длину, устанавливает поля и возвращают значения, перемещают линию, рисуют на экран. Функции-члены установки полей класса должны проверять корректность задаваемых параметров.
16. Написать класс для отдела кадров *Сотрудник* (поля: фамилия, имя, отчество, дата рождения, пол, образования, номер документа об образовании, учебное заведение выдавшее документ, дата поступления на работу, домашний адрес).
17. Создать класс записной книжки (поля: Имя, Ник, мобильный телефон, адрес электронной почты, номер ICQ).
18. Создать класс для единицы товара на складе (поля: товар, производитель, количество, дата изготовления, срок годности, поставщик, тел. поставщика, тел. производителя, цена за 1 ед.).
19. Создать класс для учета продаж (поля: товар, производитель, покупатель, количество, цена за единицу, общая стоимость).
20. Создать класс для элемента каталога музыкальных компакт дисков (поля: исполнитель, композитор, название диска, любимый трек, дата покупки, кому дан диск, количество треков, продолжительность).
21. Создать класс для элемента каталога фильмов (поля: название, режиссер, исполнитель главной роли, год выхода, кому дан на просмотр, язык звуковой дорожки, лицензионный или нет)
22. Создать базовый класс – *«Дата»*, производный класс - «*записная книжка*», включающий ФИО, телефон, дату рождения и функцию вычисления количества дней до дня рождения;

23. Создать базовый класс – время, производный класс - «расписание», включающий дисциплину, аудиторию, время начала и функцию вычисления времени до начала занятия;
24. Создать базовый класс – «окно» включающий координаты (left, top, right, bottom) и цвет окна, производный класс - «окно с текстом», включающий текст, цвет текста в окне;
25. Создать базовый класс – «полином» (массив коэффициентов), производный класс «рациональное выражение», включающий полином в числителе, полином в знаменателе.
26. Создать базовый класс «матрица», включающий матрицу, ее определитель и функцию вычисления определителя, Производный – «СЛАУ», включающий матрицу, столбец свободных членов, метод решения.
27. Создать базовый класс – «дробь», производный класс - «дробное комплексное число», включающий дробную вещественную часть, дробную мнимую часть и арифметические операции (+,-,*) над комплексными дробями;
28. Создать базовый класс «комплексное число» и производный включающий комплексное число в стандартной и экспоненциальной форме и функцию вычисления экспоненциальной формы числа.
29. Создать класс студент, имеющий имя, курс и идентификационный номер. Определить конструкторы и функцию печати. Создать производный класс - студент-дипломник, имеющий тему диплома.
30. Создать класс животное, имеющий классификацию (строка), число конечностей, число потомков. Создать производный класс - домашнее животное, имеющий кличку.
31. Создать класс машина, имеющий марку, число цилиндров, мощность. Создать производный класс - грузовик, имеющий грузоподъемность.
32. Создайте класс точка, которая имеет координаты. Создайте производный класс – эллипс (2 полуоси). Напишите виртуальный метод перемещения.

33. Создать базовый класс – компьютер (название, частота процессор, количество ядер, объем ОЗУ, постоянной памяти), производный класс – ноутбук (диагональ экрана, вес, объем батареи).
34. Создать базовый класс – мобильный телефон (Название, вес, диагональ экрана, количество встроенной памяти, наличие камеры). Производный класс – смартфон (частота процессора, количество ядер, объем ОП).
35. Создайте класс точка, которая имеет координаты. Создайте производный класс – прямоугольник (высота, ширина). Напишите виртуальный метод перемещения.
36. Создать базовый класс – дата (год, месяц, день). Производный класс – дата со временем (часы минуты).
37. Создайте базовый класс стек (на базе массива целых чисел, с максимальным размером и методами добавления и извлечения элемента) с методами добавить и извлечь элемент. Создайте класс очередь – на базе того же массива.
38. Создайте базовый класс треугольник (3 стороны, метод расчета площади). Производный класс – четырехугольник (+1 сторона и диагональ).
39. Создайте базовый класс книга (название, автор(ы), год издания, тираж, кол-во страниц). Производный класс – книга в библиотеке (инвентарный номер, кто взял).
40. Создайте базовый класс дробь (числитель, знаменатель, арифметические операции, преобразование в вещественный тип). Производный класс – число с дробной частью.

3. Ход выполнения работы

1. Загрузите IntelliJ IDEA и настройте ее, чтобы подсказка Javadoc возникала при перемещении мыши над именем класса (метода).
2. Установите плагин **simpleUMLCE** .
3. Создайте проект **Maven**.

4. Создайте класс в соответствии с вариантом индивидуального задания. Класс следует унаследовать от интерфейса. Реализуйте для него хотя бы 2 конструктора и клонирование, переопределите методы **equals**(), **hashCode**(), **toString**(). Обеспечьте инкапсуляцию данных.
5. Создайте unit-тесты для тестирования методов класса и выполните их тестирование.
6. Если обнаружены методы, не прошедшие тест, внесите необходимые изменения в код и повторите тест.
7. Сформируйте диаграмму класса.
8. Вставьте необходимые комментарии и сгенерируйте справку Javadoc.