Министерство науки и высшего образования Российской Федерации

Казанский национальный исследовательский технический университет – КАИ им. А.Н. Туполева

Институт компьютерных технологий и защиты информации

Отделение СПО ИКТЗИ «Колледж информационных технологий»

ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ

Методические указания к лабораторным работам

Казань 2020

Составитель преподаватель СПО ИКТЗИ Мингалиев Заид Зульфатович

Методические указания к лабораторным работам по дисциплине «ОСНОВЫ АЛГОРИТМИЗАЦИИ И ПРОГРАММИРОВАНИЕ» предназначены для студентов направления подготовки 09.02.07 «Информационные системы и программирование»

КНИТУ-КАИ, 2020

ОГЛАВЛЕНИЕ

ПРОЦЕСС СДАЧИ ВЫПОЛНЕННОЙ РАБОТЫ

По итогам выполнения работы студент:

1. демонстрирует преподавателю правильно работающие программы;
2. демонстрирует приобретённые знания и навыки отвечает на пару небольших вопросов преподавателя по составленной программе, возможностям её доработки;
3. демонстрирует отчет по выполненной лабораторной работе.

Итоговая оценка складывается из оценок по трем указанным составляющим.

Шаблон оформления отчета представлен в приложении 1. Требования к формированию отчета представлены в приложении 2.

ЛАБОРАТОРНАЯ РАБОТА №6.

Программирование простейших классов.

ЦЕЛЬ РАБОТЫ

Приобрести умения и практические навыки для разработки простейших классов с использованием объектно-ориентированных технологий.

ХОД РАБОТЫ

1) Понятие об объектах. Принцип инкапсуляции

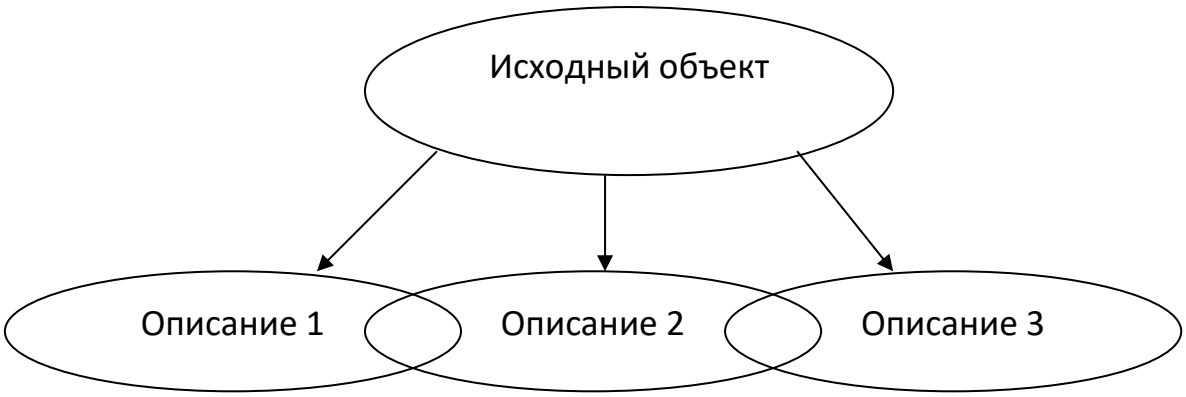
Объект – это достаточно общее и широкое понятие, которое может иметь разные трактовки. В широком смысле слова **объект – это любая сущность, имеющая некоторый набор свойств (параметров, характеристик) и обладающая некоторым поведением (функциональностью).**

Отсюда следует, что объектом может быть практически все, что угодно – **материальные** предметы (компьютер, автомобиль, человек), **логические** понятия (файл, документ, таблица, список, управляющая кнопка, окно в многооконной графической системе, форма, приложение), **математические** понятия (например, геометрические объекты типа отрезок или окружность).

Под объектом в узком смысле можно понимать некоторое **формализованное описание** рассматриваемой сущности, т.е. **модель** исходного объекта**.** Для такого описания нужен некоторый язык, например – язык программирования. В этом случае объект становится элементом языка, и именно в таком контексте он и будет рассматриваться далее.

Моделирование реальных объектов – это важнейший этап разработки объектных программ, во многом определяющий успех всего проекта. Необходимо понимать, что любая модель – это только **часть** исходного моделируемого объекта, отражающая те свойства и атрибуты объекта, которые **наиболее важны** с точки зрения решаемой задачи. Изменение постановки задачи может привести к существенному изменению модели. Особенно это характерно для сложных объектов. Вряд ли имеет смысл строить всеобъемлющую модель “на все случаи жизни”, поскольку такая модель будет очень громоздкой и практически непригодной для использования. Поэтому

один из важнейших принципов построения моделей – это принцип **абстрагирования**, т.е. выделение наиболее существенных для решаемой задачи черт исходного объекта и отбрасывание второстепенных деталей. Отсюда следует, что для одного и того же исходного объекта можно построить **разные** модели, каждая из которых будет отражать только наиболее важные в данный момент черты объекта.



Например, для исходного объекта “человек” можно построить следующие информационные модели:

- человек как студент учебного заведения
- человек как сотрудник организации
- человек как пациент учреждений здравоохранения
- человек как клиент государственных органов управления и т.д.

Из этого примера видно, что весьма часто различные модели одной и той же исходной сущности «пересекаются», имеют общие свойства (это отражено и на приведенной выше схеме): все студенты, сотрудники, клиенты имеют фамилию, дату рождения, место проживания, пол и т.д.

Как показала практика последних 10-15 лет, использование объектов при разработке **сложных** информационных систем оказалось очень удачным решением. Заказчики информационных систем, как правило, не являются специалистами в области программного обеспечения и поэтому не воспринимают (и не должны!) такие программистские термины как

переменные, подпрограммы, массивы, файлы и т.д. Гораздо легче воспринимаются **объекты** как элементы предметной области.

Разработчики на основе общения с заказчиком и на основе анализа предметной области должны построить **адекватные описания основных сущностей** решаемой задачи, перевести эти описания в необходимый **формальный** вид и создать объектную программу как **набор взаимодействующих объектов**.

Достоинством объектных программ является их четкая структуризация – каждый объект выполняет свою строго определенную задачу, а их совместная работа позволяет достичь поставленных целей.

Программный объект – это условное понятие, с которым связывается набор некоторых данных и программный код обработки этих данных

объект : данные + программный код

Объединение в рамках объекта некоторых данных и соответствующего программного кода рассматривается как проявление одного из базовых принципов объектного подхода – принципа **инкапсуляции** (**encapsulation**).

Инкапсуляция позволяет рассматривать объект в виде некоторой достаточно самостоятельной **программной единицы**, полностью отвечающей за хранение и обработку своих данных и предоставляющей посторонним пользователям четко определенный набор услуг. Однако объект это не только поставщик услуг, но чаще всего еще и потребитель услуг других объектов.

Относительно связываемых с объектом **данных** надо отметить следующее:

- каждый элемент данных часто называют **свойством** объекта (хотя есть и различия в трактовке этого термина)
- объект может содержать **любое разумное** число данных-свойств

- набор данных-свойств определяется при описании объекта и при выполнении программы изменяться не может, могут изменяться лишь **значения** свойств
- текущие **значения** свойств определяют текущее **состояние** объекта
- для хранения значений свойств необходима **память** (как для обычных переменных), и поэтому объекты программы **потребляют** оперативную память
- свойства могут иметь разные типы: **простейшие** (целочисленные, символьные, логические), **структурные** (строки, массивы, списки) и даже **объектные** (этот очень важный случай рассматривается в разделе «Взаимодействие объектов: агрегация и композиция»)
- в соответствии с принципом **инкапсуляции** элементы данных **рекомендуется** делать **недоступными** для прямого использования за пределами объекта (**закрытость** данных)

В качестве **примеров** свойств можно привести:

- объект «Студент»: фамилия, имя, дата рождения, место проживания, группа, специальность, массив оценок;
- объект «Окружность»: координаты центра, радиус, цвет.
- Относительно связываемого с объектом **программного кода** необходимо отметить следующие моменты:
- программный код разбивается на отдельные **подпрограммы**, которые принято называть **методами**
- набор методов определяет выполняемые объектом функции и тем самым реализует поведение объекта
- набор методов определяется при описании объекта и при выполнении программы уже не изменяется
- методы могут иметь **ограничения по доступности**: некоторые методы можно сделать недоступными (**закрытыми**) за пределами объекта, но

всегда должен быть определен набор **открытых** методов, образующих внешний интерфейс объекта

- вызов одного из открытых методов соответствует запросу услуги, реализуемой данным методом
- среди методов выделяют один или несколько специальных **методов-конструкторов** и иногда – один **метод-деструктор**, назначение которых рассматривается чуть дальше
- в соответствии с принципом **инкапсуляции** для доступа извне к закрытым данным могут вводиться специальные **методы доступа**

Учитывая огромную важность методов-конструкторов, рассмотрим отдельно их назначение и особенности использования.

Конструктор отвечает за **динамическое** создание нового объекта при **выполнении** программы, которое включает в себя:

- **выделение памяти**, необходимой для хранения значений свойств создаваемого объекта (совместно с операционной системой и средой поддержки выполнения программ)
- **занесение** в выделенную область **начальных значений** свойств создаваемого объекта

Важно понимать, что пока объект не создан, он не может предоставлять никакие услуги и поэтому конструктор должен вызываться **раньше** всех остальных методов. Важность конструктора определяется тем, что он **гарантированно** создает объект с какими-то **начальными** значениями свойств. Это позволяет устранить целый класс неприятных логических ошибок, таких как использование неинициализированных данных.

Для одного и того же объекта можно предусмотреть **несколько** различных конструкторов, которые **по-разному** инициализируют свойства создаваемого объекта. Начальные значения свойств часто передаются конструктору как входные параметры.

Деструктор отвечает за уничтожение объекта, т.е. освобождение памяти, выделенной объекту. В отличие от конструкторов, механизм деструкторов реализован не во всех языках: деструкторы есть в языке C++, в языке Delphi Pascal, предусмотрены (но практически не используются) в языке C# и отсутствуют в языке Java. В последних двух языках вместо деструкторов реализован специальный механизм «**сборки мусора**» (garbage collect), который способен при необходимости выполнять автоматическое освобождение памяти.

Еще одна группа очень часто используемых методов – это **методы доступа** к закрытым свойствам объекта. Введение таких методов позволяет организовать **контролируемый** доступ к внутренним данным объекта. В общем случае для закрытого свойства можно ввести **два** метода доступа:

- метод для **чтения** хранящегося в свойстве значения (часто такие методы называют **get-методами**)
- метод для **изменения** значения свойства (**set-метод**)

Набор используемых с каждым свойством методов доступа определяется при разработке объекта и позволяет для каждого свойства организовать **необходимый уровень доступа**. Если объявлены оба метода, то пользователи имеют **полный** доступ к соответствующему свойству. Если объявлен **только get-метод**, то пользователь может лишь **просматривать** значение свойства, но не изменять его. Наконец, если для свойства нет ни одного метода, то такое свойство **полностью закрыто** для пользователя.

Иногда set-методы кроме изменения значений свойств выполняют некоторую **дополнительную** работу, например – проверяют новые значения свойств.

Кроме конструкторов и методов доступа, объекты практически всегда имеют еще и некоторый набор **специфических** методов, определяющих **функциональность** объекта. Например:

- объект «Окружность»: отображение на экране, перемещение, растяжение/сжатие, вычисление длины окружности.
- объект «Студент»: посещение занятий, выполнение заданий, сдача контрольных точек, оплата обучения;

При этом каждый объект проходит определенный **жизненный цикл:**

- объект **создается** методом-конструктором
- объект **используется** другими объектами, предоставляя им свои открытые методы и неявно – закрытые данные
- объект **уничтожается** (явно деструктором или неявно механизмом сборки мусора)

Очевидно, что при работе объектной программы одновременно может существовать **множество** однотипных программных объектов (множество файлов, множество окон, множество студентов). Поэтому необходим инструмент **формального описания** таких **однотипных объектов** и в качестве такого инструмента выступает следующее важнейшее понятие объектного подхода - **класс.**

2) Классы и их описание

Класс представляет собой **формализованный способ описания однотипных объектов**, т.е. объектов с **одинаковым** набором свойств и методов**.** Именно при описании класса перечисляются свойства и реализуются методы соответствующих объектов. Разработка объектной программы начинается с описания необходимых классов.

На основе одного класса можно создать **любое разумное** (все ресурсы вычислительной системы конечны!) число объектов, называемых **экземплярами** этого класса. Все используемые в программе объекты должны быть экземплярами некоторых классов, стандартных или собственных. Соответствие между понятиями «объект» и «класс» аналогично соответствию между понятиями «переменная» и «тип данных».

Правила описания классов различны для разных языков, тем не менее можно отметить ряд **общих** моментов. После этого будут приведены особенности описания классов для конкретных языков.

Описание класса включает в себя:

- **заголовок** класса, включающий специальную директиву **class** и имя класса; практически всегда в заголовке класса задается дополнительная информация о классе;
- **тело** класса, содержащее перечень **свойств** (полей данных), **заголовки** методов и их **программную реализацию** (не всегда).

В классах допускается объявлять так называемые **абстрактные** методы, у которых есть **только заголовок** (имя и параметры), но **нет программной реализации**. Целесообразность использования таких конструкций будет обоснована при рассмотрении принципа наследования. Если класс содержит **хотя бы один** абстрактный метод, он сам считается **абстрактным**. Важная особенность абстрактных классов состоит в том, что **объекты-экземпляры на их основе создавать нельзя**! Абстрактные классы используются для описания абстрактных понятий.

Общие правила описания свойств:

- каждое свойство объявляется как обычная переменная, т.е. для нее обязательно задается **имя** и **тип** (простейший, структурный или объектный)
- имена всех свойств в классе должны быть различными
- свойства рекомендуется объявлять **закрытыми** с помощью специальной директивы **private** (такие свойства доступны для прямого использования только внутри методов данного класса); открытые свойства (если необходимо нарушить принцип инкапсуляции) объявляются с помощью директивы **public**
- свойство может быть объявлено **статическим** или **классовым;** такое свойство является характеристикой **класса** в целом, и поэтому все

объекты-экземпляры класса будут иметь одно и то же значение этого свойства; такие свойства можно использовать **БЕЗ** создания объектов!

Общие правила описания методов:

- метод оформляется как обычная подпрограмма, с указанием **имени**, формальных **параметров** (если необходимо) и типа возвращаемого значения (если метод должен иметь функциональную форму)
- допускается объявлять **несколько** методов с **одним и тем же** именем, но **разными** наборами формальных параметров (так называемая **перегрузка** методов, **overloading**); наборы параметров должны отличаться либо их числом, либо типами параметров
- каждый метод должен быть объявлен либо как **открытый** (директива **public**), либо как **закрытый** (**private**); закрытые методы могут использоваться только другими методами данного класса
- методы-конструкторы объявляются по своим правилам (см. дальше)
- метод может быть объявлен **статическим**, **классовым**; такой метод можно вызывать **БЕЗ** создания объектов!
- **абстрактные** методы объявляются **с помощью** директивы **abstract**
- методы доступа для **чтения** значений закрытых свойств (Get-методы) **рекомендуется** именовать с префиксом **Get** (например GetColor, GetSize) и оформлять как **функцию** без параметров, возвращающую тип соответствующего свойства
- методы доступа для **изменения** значений закрытых свойств (Set-методы) **рекомендуется** именовать с префиксом **Set** (например SetColor, SetSize) и оформлять как процедуру (void-функцию) с одним **входным** параметром

Теперь после рассмотрения общих моментов в описании классов можно перейти к конкретным правилам и примерам описания классов в различных объектных языках.

3) Класс на C#

Общая структура класса (заголовок и тело):

```
1  class    ИмяКласса
2  { набор свойств;
3  набор методов;
4  };
```

Свойства и методы перечисляются в произвольном порядке, причем **каждый** элемент класса должен иметь **свою** директиву **public** или **private**

Все конструкторы имеют **одно и то же имя**, **совпадающее** (с точностью до регистра!) с **именем** класса. Отсюда следует, что при объявлении в классе более одного конструктора, неизбежно возникает ситуация **перегрузки**, и поэтому конструкторы **должны** иметь **разные** наборы параметров.

Программная реализация неабстрактных методов приводится **сразу за заголовком** метода, т.е. нет никаких вынесенных в другое место фрагментов. Это имеет как **положительные** (вся структура класса собрана в одном месте), так и **отрицательные** стороны (описание класса с объемными методами может занимать несколько страниц).

Set-методы оформляются как void-функции, принимающие необходимые входные параметры. Get-методы оформляются как функции, возвращающие значения запрашиваемого свойства.

Несколько классов можно собрать одном **пространстве имен** (**namespace**, **C#**). Для таких классов **отключается** механизм ограничения доступа к элементам класса.

Статические (классовые) свойства и методы задаются директивой **static.**

Пример описания класса окружностей (некоторые методы пропущены):

```
1  class Circle
2  { private int x;
3  private int y;
4  private int r;
5  public Circle ( ) {x = 0; y = 0; r = 100 ;}
```

```
6   public Circle (int ax, int ay, int ar) {x = ax; y = ay;
    r = ar ;}
7   public int GetX ( ) {return x ;}
8   public void MoveTo (int ax, int ay)
9   {«стереть»; x = x + ax; y = y + ay; Show ( ); }
10  };   // конец описания класса
```

4) Класс на C++

Заголовок класса включает имя класса и директиву **class.** Все конструкторы класса имеют **одно и то же имя, совпадающее с именем класса.** Деструктор только один, его имя также совпадает с именем класса, но перед именем ставится символ ~ (тильда, волнистая черта). Директивы **public** и **private** используются так же, как и в языке DP. Программную реализацию методов можно либо вынести за пределы класса, либо вставить сразу за заголовком метода. Для вынесенной реализации в заголовке обязательно надо сначала указать имя класса. Классовые (статические) свойства и методы объявляются с помощью директивы **static.**

Пример для класса окружностей (некоторые методы пропущены):

```
1   class  Circle
2   { private:
3   int  x;  int  y;  int  r;
4   public:
5   Circle ( ); // конструктор без параметров
6   Circle (int ax, int ay, int ar);  // конструктор с
    тремя параметрами
7   Circle (int,  int);                          //
    конструктор с двумя параметрами
8   int GetX ( )  { return  x};        // встроенная
    реализация метода
```

9	**void** SetXY (int ax, int ay) {x = ax; y = ay}; // аналогично
10	**void** MoveTo (int ax, int ay); // реализация вынесена
11	**void** Show ( ); // аналогично
12	~Circle( ); // деструктор
13	}; // конец тела класса

Программная реализация некоторых методов **за пределами** класса:

1	Circle::Circle ( ) // реализация конструктора без параметров
2	{x = 0; y = 0; r = 0};
3	Circle::MoveTo (int ax, int ay) // реализация метода перемещения
4	{«стереть»; SetXY(ax, ay); Show ( ); };

5) Задание на лабораторную работу

В ходе выполнения контрольной работы также необходимо, определив свой вариант, выполнить задания, перечисленные в таблице.

	Первое задание	Второе задание
1	1-1	2-15
2	1-2	2-14
3	1-3	2-13
4	1-4	2-12
5	1-5	2-11
6	1-6	2-10
7	1-7	2-9
8	1-8	2-8
9	1-9	2-7
10	1-10	2-6
11	1-11	2-5
12	1-12	2-4
13	1-13	2-3
14	1-14	2-2
15	1-15	2-1
16	1-16	2-15

17	1-17	2-14
18	1-18	2-13
19	1-19	2-12
20	1-20	2-11

Для всех рассматриваемых ниже заданий разработать класс, содержащий два члена (назовем их first, second), и следующие методы:

- ввод с клавиатуры Read;
- вывод на экран Display;
- метод, указанный в задании.

Таблица 1 «Первое задание»

1-1	Элемент ai арифметической прогрессии вычисляется по формуле ai + 1 = ai + d, i = 0, 1, 2, ... Поле first – вещественное число, первый элемент прогрессии a0; поле second – разность прогрессии d. Определить метод element_i() – для вычисления заданного элемента прогрессии.
1-2	Поле first – вещественное число; поле second – вещественное число, показатель степени. Реализовать метод power( ) – возведение числа first в степень second.
1-3	Поле first – целое положительное число, числитель; поле second – целое положительное число, знаменатель. Реализовать метод ipart( ) – выделение целой части дроби first/second.
1-4	Поле first – целое положительное число, номинал купюры; номинал может принимать значения 1, 2, 5, 10, 50, 100, 500, 1000, 5000. Поле second – целое положительное число, количество купюр данного достоинства. Реализовать метод summa( ) –вычисление денежной суммы.
1-5	Поле first – вещественное положительное число, цена товара; поле second – целое положительное число, количество единиц товара. Реализовать метод cost( ) – вычисление стоимости товара.

1-6	Поле first – целое положительное число, калорийность 100 г продукта; поле second – вещественное положительное число, масса продукта в килограммах. Реализовать метод Power( ) –вычисление общей калорийности продукта.
1-7	Поле first–вещественное число, левая граница диапазона; поле second–вещественное число, правая граница диапазона. Реализовать метод rangecheck( ) –проверку заданного числа на принадлежность диапазону.
1-8	Поле first–целое число, левая граница диапазона, включается в диапазон; поле second–целое число, правая граница диапазона, не включается в диапазон. Пара чисел представляет полуоткрытый интервал [first, second). Реализовать метод rangecheck( ) –проверку заданного числа на принадлежность диапазону.
1-9	Поле first – целое положительное число, часы; поле second – целое положительное число, минуты. Реализовать метод minutes( ) – приведение времени в минуты.
1-10	Линейное уравнение y = Ax + B. Поле first–вещественное число, коэффициент A; поле second–вещественное число, коэффициент B. Реализовать метод function( ) – вычисление для заданного x значение функции y.
1-11	Линейное уравнение y = Ax + B. Поле first – вещественное число, коэффициент A; поле second – вещественное число, коэффициент B. Реализовать метод function( ) – вычисление корня линейного уравнения.
1-12	Поле first – вещественное число, координата x точки на плоскости; поле second – вещественное число, координата y точки на плоскости. Реализовать метод distance( ) –расстояние точки от начала координат.

1-13	Поле first – вещественное число, катет a прямоугольного треугольника; поле second – вещественное число, катет b прямоугольного треугольника. Реализовать метод hypotenuse() – вычисление гипотенузы.
1-14	Поле first–вещественное положительное число, оклад; поле second –целое положительное число, количество отработанных дней в месяце. Реализовать метод summa( ) – вычисление начисленной зарплаты для заданного месяца: оклад / дни_месяца * отработанные_дни.
1-15	Поле first – целое положительное число, продолжительность телефонного разговора в минутах; поле second – вещественно положительное число, стоимость одной минуты в рублях. Реализовать метод cost( ) – вычисление общей стоимости разговора.
1-16	Поле first – вещественное число, целая часть числа; поле second – положительное вещественное число, дробная часть числа. Реализовать метод multiply( ) – умножение на произвольное вещественное число типа double.
1-17	Поле first – целое положительное число, числитель; поле second – целое положительное число, знаменатель. Реализовать метод nesokr( ) – приведение дроби first/second к несократимому виду.
1-18	Поле first – целое число, целая часть числа; поле second – положительное целое число, дробная часть числа. Реализовать метод multiply( ) – умножение на произвольное целое число типа int.
1-19	Число сочетаний из n объектов по k объектов (k<n) вычисляется по формуле: C(n,k)=n!/((n –k)!*k!). Поле first – целое положительное число, k; поле second – положительное целое число, n. Реализовать метод combination( ) – вычисление C(n,k).

1-20	Элемент aj геометрической прогрессии вычисляется по формуле aj= a0*r, j = 0,1,2, ... Поле first – вещественное число, первый элемент прогрессии a0; поле second – знаменатель прогрессии, r. Определить метод elementj( ) –для вычисления заданного элемента прогрессии.

Таблица 2 «Второе задание»

2-1	Создать класс EngMer для работы с английскими мерами длины: фунтами и дюймами, при этом учтем, что 1 фунт = 12 дюймов. Длина объекта будет задаваться парой чисел (фунты и дюймы), нужно реализовать: сложение и вычитание длин, умножение и деление длин, сравнение длин.
2-2	Создать класс vector3D, задаваемый тройкой координат. Обязательно должны быть реализованы: сложение и вычитание векторов, скалярное произведение векторов, умножение на скаляр, сравнение векторов, вычисление длины вектора, сравнение длины векторов.
2-3	Создать класс EngMoney для работы с устаревшей денежной системой Великобритании. В ней использовались фунты, шиллинги и пенсы. При этом: 1 фунт = 20 шиллингов, 1 шиллинг =12 пенсов. Денежные суммы будут задаваться в фунтах, шиллингах и пенсах и результат выдаваться также в этих величинах. Должны быть реализованы: сложение и вычитание, умножение и деление, сравнение сумм.
2-4	Создать класс Money для работы с денежными суммами. Число должно быть представлено двумя полями: типа long int для рублей и типа int – для копеек. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой. Реализовать

	сложение, вычитание, деление сумм, деление суммы на дробное число, умножение на дробное число и операцию сравнения.
2-5	Создать класс Triangle для представления треугольника. Поля данных должны включать углы и стороны. Требуется реализовать операции: получения и изменения полей данных, вычисления площади, вычисления периметра, вычисления высот, а также определения вида прямо-угольника (равносторонний, равнобедренный или прямоугольный).
2-6	Создать класс Angle для работы с углами на плоскости, задаваемыми величинами в градусах и минутах. Обязательно должны быть реализованы: перевод в радианы, приведение к диапазону 0–360, увеличение и уменьшение угла на заданную величину, получение синуса, сравнение углов.
2-7	Создать класс Point для работы с точками на плоскости. координаты точки – декартовы. Обязательно должны быть реализованы: перемещение точки по оси X, перемещение по оси Y, определение расстояния до начала координат, расстояние между двумя точками.
2-8	Рациональная (несократимая) дробь представляется парой целых чисел (a,b), где a–числитель, b–знаменатель. Создать класс Rational для работы с рациональными дробями. Обязательно должны быть реализованы операции: сложения, вычитания, умножения, деления и сравнения дробей, причем результат должен приводиться в виде несократимой дроби (т.е. результат нужно сокращать).
2-9	Создать класс Date для работы с датами в формате «год.месяц.день». Дата представляется структурой с тремя полями типа unsigned int: для года, месяца и дня. Класс должен включать операции: вычисление даты через заданное количество дней от указанной, вычитание заданного количества дней из даты,

	определение високосного года, присвоение и получение отдельных частей (год, месяц, день), сравнение дат (равно, до, после), вычисление количества дней между датами.
2-10	Создать класс Time для работы со временем в формате «час:минута:секунда». Обязательными операциями являются: вычисление разницы между двумя моментами времени в секундах, сложение времени и заданного количества секунд, вычитание из времени заданного количества секунд, сравнение моментов времени, перевод в секунды, перевод в минуты (с округлением до целой минуты)
2-11	В морской навигации координаты точки измеряются в градусах и минутах широты и долготы. Один градус равен 60 минутам (ранее минуту делили на 60 секунд, но сейчас минуту делят на обычные десятичные доли). Долгота измеряется от 0 до 180°восточнее (E) или западнее (W) Гринвича. Широта принимает значения от 0 до 90°севернее (N) или южнее (S) экватора. Создать класс Koord, включающий следующие три поля: типа int для числа градусов, типа float для числа минут и типа char для указания направления (N,S, W или E). Объект этого класса может содержать значение как широты, так и долготы. Предусмотреть ввод координат точки, указания полушария для указанной точки, вычисления расстояния между двумя точками. Учитывать, что по широте 1°равен 111 км, а по долготе 1°равен 111 км * cos(широты).
2-12	Реализовать класс Account, представляющий собой банковский счет. В классе должны быть реализованы 4 поля: фамилия владельца, номер счета, процент начисления и сумма в рублях. Необходимо выполнять следующие операции: сменить владельца счета, снять некоторую сумму со счета, положить деньги на счет,

	начислить проценты, перевести сумму в доллары, перевести сумму в евро, получить сумму прописью (преобразовать в числительное).
2-13	Номиналы российских рублей могут принимать значения 1, 2, 5, 10, 50, 100, 500, 1000, 5000. Копейки представить как 0,01 (1 копейка), 0,05 (5 копеек), 0,1 (10 копеек), 0,5 (50 копеек). Создать класс Money для работы с денежными суммами. Сумма должна быть представлена полями-номиналами, значениями которых должны быть количества купюр данного достоинства. Реализовать сложение сумм, вычитание сумм, деление сумм, деление суммы на дробное число, умножение на дробное число и операцию сравнения. Дробная часть (копейки) при выводе на экран должна быть отделена от целой части запятой.
2-14	Создать класс Goods(товар). В классе должны быть представлены поля: наименование товара, дата оформления, цена товара, количество единиц товара, номер накладной, по которой товар поступил на склад. Реализовать методы изменения цены товара, изменения количества товара (увеличения и уменьшения), вычисления стоимости товара. Должен быть метод для отображения стоимости товара в виде строки.
2-15	Создать класс Payment(зарплата). В классе должны быть представлены поля: фамилия-имя-отчество, оклад, год поступления на работу, процент надбавки, подоходный налог, количество отработанных дней в месяце, количество рабочих дней в месяце, начисленная и удержанная сумма. Реализовать методы: вычисления начисленной суммы, вычисления удержанной суммы, вычисления суммы, выдававаемой на руки, вычисления стажа. Стаж вычисляется как полное количество лет, прошедших от года поступления на работу, до текущего года. Начисления представляют собой сумму, начисленную за отработанные дни, и

	надбавки. Удержания представляют собой отчисления в пенсионный фонд (1% от начисленной суммы) и подоходный налог (13%).

6) Контрольные вопросы

1. Что такое объект?
2. Что такое класс?
3. Приведите пример классов.
4. В чем заключается принцип инкапсуляции?
5. В чем заключается принцип абстрагирования?
6. Какие есть правила для описания свойств и методов?
7. Для чего используются get/set методы?
8. Что такое конструктор класса?