**Space (1)**

Давай напишем новую компьютерную игрушку.  
У нас будет космический корабль, который стреляет ракетами в НЛО.  
А НЛО в свою очередь сбрасывает на корабль бомбы.  
Ну и, конечно, всё дело происходит в космосе.

Нам понадобятся классы: Space(*космос*), SpaceShip(*космический корабль*) и Ufo(*НЛО*).  
Создай классы: Space, SpaceShip, Ufo.  
Добавь метод main в класс Space.

# Требования: 1. Создай в отдельном файле класс Space. 2. Создай в отдельном файле класс SpaceShip. 3. Создай в отдельном файле класс Ufo. 4. Добавь метод main в класс Space.

### Space (2)

У нас будут бомбы и ракеты.  
Значит нам нужны классы **Bomb**(бомба) и Rocket(ракета).  
Создай их.

У наших объектов будет много общего.  
Они будут перемещаться по космосу и отрисовываться.  
Значит у них у всех будут координаты и размер.  
А еще методы move() - для перемещения и draw() для отрисовки.

Есть интересное предложение: давай введем один базовый класс для всех объектов.  
Пусть это будет класс BaseObject.  
А классы **Ufo**, **SpaceShip**, **Rocket**, **Bomb** от него наследуются.  
Создай абстрактный класс **BaseObject** и добавь его родителем к классам **Ufo**, **SpaceShip**, **Rocket**, Bomb.

Еще нам понадобится класс **Canvas**.  
Он будет ответственным за "***отрисовку***" объектов.  
С его помощью они будут отрисовывать себя.  
Вернее даже на нем.  
Создай и этот класс.

### Space (3)

Для чего нам нужен класс **Space**?  
Чтобы хранить в себе все объекты и управлять их взаимодействием.  
А какие параметры должны у него быть?  
width (ширина), height (высота).  
А еще?  
а) ship (космический корабль),  
б) список для хранения всех **НЛО** - ufos (*ArrayList<Ufo>*),  
в) список для хранения всех **ракет** - **rockets** (ArrayList<Rocket>),  
г) список для хранения всех **бомб** - **bombs** (*ArrayList<Bomb*>).

**Задание:**  
Добавь все эти переменные к классу **Space**.  
Инициализируй коллекции.  
И не забудь добавить переменным **getter'ы**, а для ship еще и **setter**!

А что должен содержать конструктор?  
Достаточно будет width и height.

Требования:  
1. В классе Space создай поле width. Добавь для него getter.  
2. В классе Space создай поле height. Добавь для него getter.  
3. В классе Space создай поле ship. Добавь для него getter и setter.  
4. В классе Space создай поле ufos. Добавь для него getter.  
5. В классе Space создай поле rockets. Добавь для него getter.  
6. В классе Space создай поле bombs. Добавь для него getter.  
7. В классе Space создай конструктор, который будет инициализировать width и height.  
8. Инициализируй поля с коллекциями.

### Space (4)

Чего не хватает классу **Space**?  
Правильно - методов run() и draw().  
run управляет всей логикой игры, если ты помнишь.  
А draw отвечает за отрисовку очередного "***кадра***".

А еще нам пригодится метод sleep(int ms).  
Создай их.

Требования:  
1. В классе Space создай метод run().  
2. В классе Space создай метод draw().  
3. В классе Space создай метод sleep(int ms).

### Space (5)

Теперь перейдем к классу **BaseObject**.  
Я хочу сделать несколько предложений.

**Во-первых.** Для простоты, считать все объекты у нас в космосе круглыми.  
Нет, отрисовывать их мы будем фигурными, как и раньше.  
А вот при расчетах их взаимодействия исходить из того, что они круглые.  
Так - гораздо проще.

**Во-вторых.** Пусть координаты объектов и радиус будут вещественными числами.  
Это придаст плавность движениям и точность всем вычислениям.  
А при отрисовке мы будем их округлять.

**Надо:**  
а) Добавь в класс **BaseObject** переменные x (double), y (double), radius (double), **геттеры** и **сеттеры** для них.  
б) Добавить логическую переменную isAlive (жив объект или уже нет).  
в) Добавить геттер (isAlive()-метод для isAlive-переменной).  
г) Добавить конструктор BaseObject(double **x**, double **y**, double **radius**).  
д) Проследить, чтобы в конструкторе isAlive устанавливался в **true** (мертворожденные нам ни к чему).  
е) Надо пройтись по всем классам-наследникам и поправить у них конструкторы.  
Если вы пользуетесь **Intellij IDEA** - Alt+Insert вам в помощь.

Требования:  
1. В классе BaseObject создай поле x (double). Добавь для него getter и setter.  
2. В классе BaseObject создай поле y (double). Добавь для него getter и setter.  
3. В классе BaseObject создай поле radius (double). Добавь для него getter и setter.  
4. В классе BaseObject создай поле isAlive (boolean). Добавь для него getter.  
5. В классе BaseObject создай конструктор BaseObject(double x, double y, double radius). Поле isAlive должно устанавливаться в true.  
6. Создай аналогичные конструкторы во всех классах, которые наследуются от BaseObject.

### Space (6)

Но и это еще не все.  
Классу **BaseObject** нужны еще методы.  
Пока это будут пустые методы draw() и move().  
Классы-наследники должны будут переопределить их у себя и реализовать необходимую функциональность.

Еще добавь метод die() - объект умирает (isAlive=false)

А еще нам нужно будет определять попала бомба в корабль или ракета в **НЛО**.  
Это будем делать так:  
Создадим специальный метод: public boolean isIntersect(BaseObject **o**)  
Он будет определять - "***пересеклись***" объекты или нет. Если пересеклись - возвращать **true**, если нет - **false**.

Т.к. объекты мы условно считаем кругами, то предлагаю такую формулу взаимодействия:  
eсли центр круга одного объекта попал в круг другого, то будем считать, что они столкнулись.  
**Или еще проще:**  
дистанция\_между\_объектами < max (радиус\_первого\_объекта, радиус\_второго\_объекта).

Требования:  
1. В классе BaseObject создай пустой метод draw().  
2. В классе BaseObject создай пустой метод move().  
3. В классе BaseObject создай метод die(), который присваивает полю isAlive значение false.  
4. В классе BaseObject создай метод isIntersect(BaseObject o), который возвращает boolean.  
5. Реализуй метод isIntersect(BaseObject o). В случае если объекты столкнулись, нужно вернуть true, иначе – false

### Space (7)

Теперь займемся классом **Canvas**.  
Он у нас будет содержать матрицу, куда мы будем рисовать.  
У матрицы есть ширина и высота.  
А еще будем в ней хранить не **числа**(int), а **символы** (char).

**Надо:**  
а) Добавить в класс две переменные width и **height**;  
б) Добавить в класс переменную matrix (char[][]);  
в) Добавь **геттеры** для них;  
г) В конструкторе проинициализируй **матрицу**.

Требования:  
1. В классе Canvas создай поле width. Добавь для него getter.  
2. В классе Canvas создай поле height. Добавь для него getter.  
3. В классе Canvas создай поле matrix (char[][]). Добавь для него getter.  
4. В классе Canvas создай конструктор Canvas(int width, int height). Инициализируй поля width и height.  
5. Инициализируй в конструкторе поле matrix (char[height][width]).

### Space (8)

Что мы будем делать с **Canvas**?  
Мы будем рисовать на нем (в его матрице).

**Поэтому нам понадобятся два метода:**  
public void setPoint(double x, double y, char c),  
public void drawMatrix(double x, double y, int[][] matrix, char c).

Первый метод - setPoint будет "ставить точку в координатах x,y цветом c".  
**В методе надо:**  
а) округлить x и y до целых чисел,  
б) занести в matrix[y][x] значение **с**,  
в) ничего не делать, если x**<0** или y**<0** или y**>**matrix.length или x**>**matrix[0].length.

Второй метод - drawMatrix копирует переданную ему картинку (матрицу) в матрицу **Canvas**.  
И не просто копирует, а начиная с координат x, y.

**В методе надо:**  
а) с помощью двух вложенных циклов пройтись по всем ячейкам переданной картинки,  
б) если значение ячейки matrix[i][j] не равно **0**, то покрасить в матрице объекта Canvas точку (x+j, y+i) в цвет **c**:  
setPoint(x+j, y+i, c)

Требования:  
1. В классе Canvas создай метод public void setPoint(double x, double y, char c).  
2. Метод setPoint должен заносить в матрицу по координатам x, y символ с.  
3. В классе Canvas создай метод public void drawMatrix(double x, double y, int[][] matrix, char c).  
4. Метод drawMatrix, начиная с координат x, y, должен заполнять полотно символами с, согласно переданной в метод матрицей matrix.

### Space (9)

Еще **Canvas** понадобится два метода, напиши их.  
а) метод clear();  
Этот метод будет очищать матрицу, чтобы на ней снова можно было рисовать.  
Например заменить все символы матрицы на пробелы.

б) метод print();  
Этот метод отрисовывает матрицу на экран.  
Тут уже ты должен сам разобраться: вывести набор символов не так уж и сложно.  
Не забудь добавить пару пустых строк в конце, чтобы матрицы выведенные в разное время не слипались.

Требования:  
1. В классе Canvas создай метод clear().  
2. Метод clear() должен очищать матрицу.  
3. В классе Canvas создай метод print().  
4. Метод print() должен отрисовыть матрицу в консоль.

### Space (10)

Теперь напишем класс **Bomb**. Тут все просто.  
**Bomb** унаследован от **BaseObject**.

**Надо:**  
а) изменить конструктор:  
Конструктор Bomb должен выглядеть примерно так:  
public Bomb(double x, double y)  
{  
super(x, y, 1);  
}

x и y переданные в конструктор Bomb мы передаем дальше в конструктор BaseObject с помощью super, где так же указываем радиус "бомбы" равный **1**.

б) написать метод move():  
тут все просто - бомба падает вниз - просто увеличиваем y на **1**.

в) метод draw(Canvas canvas):  
тут тоже не очень сложная логика.  
Давай просто ставить точку с координатами (x,y) и "***цветом***" **B**:  
canvas.setPoint(x,y,'B').

г) Создай в классе **Space** нашу игру public static Space game.

Требования:  
1. В классе Bomb конструктор должен принимать координаты, а радиус выставлять всегда равным 1.  
2. В классе Bomb напиши метод move(), который увеличивает координату y на единицу.  
3. В классе Bomb напиши метод draw(Canvas canvas), который для canvas будет ставить символ 'B' в координатах x, y.  
4. В классе Space создай поле public static Space game.

### Space (11)

Теперь напишем класс **Rocket**.  
Практически совпадает с классом **Bomb**.  
**Только:**  
а) Ракета летит вверх (т.е. y уменьшается на 1);  
б) Рисуем не букву "***B***", а букву "***R***".

Требования:  
1. В классе Rocket конструктор должен принимать координаты, а радиус выставлять всегда равным 1.  
2. В классе Rocket напиши метод move(), который уменьшает координату y на единицу.  
3. В классе Rocket напиши метод draw(Canvas canvas), который для canvas будет ставить символ 'R' в координатах x, y.

### Space (12)

Теперь займемся кораблем.  
Для сложности сделаем так: если пользователь нажал кнопку влево, то корабль начинает все время двигаться влево.  
Если нажал кнопку вправо - все время в право до упора.  
Поэтому заведем специальную переменную dx, которая будет хранить направление движения корабля.  
если dx = 1, корабль каждый ход идет на **1** **вправо**,  
если dx = -1, корабль каждый ход идет на **1** **влево**.

**Надо:**  
а) добавь конструктор, можно такой:  
Радиус корабля будет равен **3**. Корабль большой - это вам не ракета и не бомба.  
public SpaceShip(double x, double y)  
{  
super(x, y, 3);  
}  
б) добавь в класс **SpaceShip** переменную dx (*double*, по умолчанию равна***0***);  
в) метод moveLeft(), устанавливает dx равной **-1**;  
г) метод moveRight(), устанавливает dx равной **1**.

Требования:  
1. В классе SpaceShip конструктор должен принимать координаты x и y, а радиус выставлять всегда равный 3.  
2. В классе SpaceShip добавь поле dx (double).  
3. В классе SpaceShip напиши метод moveLeft(), который устанавливает поле dx равным -1.  
4. В классе SpaceShip напиши метод moveRight(), который устанавливает поле dx равным 1.

### Space (13)

Теперь перейдем к методам draw, move.  
В методе move() надо:  
а) увеличить x на dx;  
б) проверить, не вылез ли корабль за границы космоса [0, Space.game.getWidth()]. Используй метод checkBorders.  
Учти, что ширина корабля равна двум его радиусам.

Метод draw я напишу сам - просто объяви пустой метод.

Еще нам понадобится метод fire(), ведь корабль умеет стрелять.  
Этот метод вызывается, когда надо произвести выстрел.

**В этом методе надо:**  
а) создать две ракеты;  
б) установить им координаты левого края корабля и правого края корабля (пушки находятся на расстоянии 2 от центра корабля);  
в) добавить эти ракеты в список ракет объекта game.

Его можно получить так:  
Space.game.getRockets()

Требования:  
1. В классе SpaceShip напиши метод move(), который увеличивает поле x на значение dx и проверяет, не вылез ли корабль за границы космоса.  
2. В классе SpaceShip добавь пустой метод draw(Canvas canvas).  
3. В классе SpaceShip напиши метод fire(), который создает две ракеты с координатами левого и правого края корабля и добавляет их в поле rockets у объекта game.

### Space (14)

Теперь напиши класс **Ufo**.  
Он чуть сложнее класса **Bomb** и проще **SpaceShip**.  
Тебе понадобятся методы draw(), move(), fire().  
Также не забудь о конструкторе. Типы параметров double, а радиус **ufo** будет равен **3**.

Метод draw() сделай по аналогии с **SpaceShip**. Рисовать **UFO** будем символом '***U***'.  
И добавь матрицу, по аналогии с **SpaceShip**.

Метод move() надо сделать так:  
а) корабль перемещается по случайной траектории;

**Подсказка:**  
double dx = Math.random() \* 2 - 1; //-1..1  
double dy = Math.random() \* 2 - 1; //-1..1

б) корабль не опускается в нижнюю половину экрана;  
y <= height/2

в) С вероятностью **10%** корабль должен стрелять - вызывать метод fire.

Метод fire():  
Корабль сбрасывает одну бомбу по середине.  
Сделать аналогично классу SpaceShip.

Требования:  
1. В классе Ufo конструктор должен принимать координаты x и y, а радиус выставлять всегда равным 3.  
2. В классе Ufo добавь поле с матрицей объекта по аналогии с классом SpaceShip.  
3. В классе Ufo напиши метод draw(), который должен отрисовывать объект по аналогии с класcом SpaceShip.  
4. В классе Ufo напиши метод move(), который изменяет координаты объекта согласно заданию.  
5. В классе Ufo напиши метод fire(), который создает бомбу под объектом по середине и добавляет ее в список Space.bombs.  
6. В классе Ufo в методе move() добавь вызов метода fire() c вероятностью 10%.