Арканоид(1)

Давай сегодня напишем игру **Арканоид**.

В оригинале это выглядит примерно так http://www.youtube.com/watch?v=Th-Z6QQ5AOQ  
Для начала мы напишем **упрощенную** **версию** этой игры.  
У нас будут кубики, шарик и летающая «***подставка***«, которая не дает шарику упасть.

Поэтому нам понадобятся классы:  
а) **Arkanoid** — класс в котором происходят основные действия. Также в нем будет наша точка входа в программу, метод main.  
б) **Ball** — **шарик**.  
в) **Brick** — «***кирпичи***«, которые мы сбиваем шариком.  
г) **Stand** — **летающая** **подставка**.

Создай их, и не забудь о методе public static void main (String[] **args**) в класе **Arkanoid**.

Требования:  
1. Класс Arkanoid должен быть создан в отдельном файле.  
2. Класс Ball должен быть создан в отдельном файле.  
3. Класс Brick должен быть создан в отдельном файле.  
4. Класс Stand должен быть создан в отдельном файле.  
5. В классе Arkanoid должен быть создан метод public static void main (String[] args).

**Арканоид(2)**

Главному классу (Arkanoid) нужно будет хранить информацию о размерах поля, где будут происходить все действия.

Поэтому:  
а) добавь в класс Arkanoid два приватных поля width(ширина) и height(высота) типа int  
б) создай для них публичные геттеры и сеттеры

### Арканоид(3)

Пожалуй, пора добавить конструктор в класс **Arkanoid**.  
Конструктор должен принимать два параметра типа int (width и height) и корректно устанавливать значения соответствующих полей класса.

### Арканоид(4)

Также классу **Arkanoid** нужно будет хранить ссылку на шарик (**Ball**), «***подставку***» **Stand** и список «***кирпичей***«.

**Надо:**  
а) создать в классе **Arkanoid** два private поля: ball типа **Ball** и stand типа Stand  
б) добавить private поле bricks типа **List**<Brick>  
в) добавь для них публичные **геттеры** и **сеттеры**

### Арканоид(5)

Чего еще не хватает классу **Arkanoid**?  
Во-первых ему нужен метод run(), в котором будет описана основная логика программы.  
Еще нужен метод move() — который будет двигать на один шаг все объекты требующие движения.  
Создай методы run() и move().

Еще нам понадобится **НЕ** приватная статическая переменная game типа **Arkanoid**, которая будет хранить ссылку на созданный экземпляр класса **Arkanoid**.

**Выглядеть это должно примерно так:**  
static Arkanoid game;

### Арканоид(6)

У нас будут «***кирпичи***«, «***шарик***» и «***подставка***» и у них будет много общего.  
Они будут перемещаться по полю 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 все объектов.  
Пусть это будет класс **BaseObject**.  
А классы **Ball**, **Stand**, **Brick** от него наследуются.

Создай класс **BaseObject** и добавь его родителем к классам **Ball**, **Stand**, **Brick**

Еще нам понадобится класс **Canvas**  
Он будет ответственным за «***отрисовку***» объектов.  
С помощью его они будут отрисовывать себя.  
Вернее даже на нем, но детали я сообщу позднее.

**Создай и этот класс.**

**Арканоид(7)**

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

Во-первых для простоты считать все объекты у нас будут круглыми.  
Нет, отрисовывать их мы будем фигурными, как и раньше.  
А вот при расчетах из 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.

Итак:  
а) Добавь в класс BaseObject приватные поля x (double), y (double), radius (double).  
б) Добавь геттеры и сеттеры.  
в) Добавь конструктор BaseObject(double x, double y, double radius).  
г) Пройдись по все классам-наследникам и поправь у них конструкторы.

Если вы пользуйтесь Intellij IDEA — Alt+Insert вам в помощь.

Требования:  
1. В классе BaseObject должно быть создано приватное поле x типа double.  
2. В классе BaseObject должно быть создано приватное поле y типа double.  
3. В классе BaseObject должно быть создано приватное поле radius типа double.  
4. В классе BaseObject должен быть создан корректно работающий конструктор с тремя параметрами типа double (x, y, radius).  
5. В классе Ball должен быть создан корректно работающий конструктор с тремя параметрами типа double (x, y, radius).  
6. В классе Brick должен быть создан корректно работающий конструктор с тремя параметрами типа double (x, y, radius).  
7. В классе Stand должен быть создан корректно работающий конструктор с тремя параметрами типа double (x, y, radius).  
8. В классе BaseObject должен быть создан публичный геттер для поля x.  
9. В классе BaseObject должен быть создан публичный геттер для поля y.  
10. В классе BaseObject должен быть создан публичный геттер для поля radius.  
11. В классе BaseObject должен быть создан публичный сеттер для поля x.  
12. В классе BaseObject должен быть создан публичный сеттер для поля y.  
13. В классе BaseObject должен быть создан публичный сеттер для поля radius.

### Арканоид(8)

Но и это еще не все.  
Классу BaseObject нужны еще методы.  
Пока это будут пустые методы draw(Canvas canvas) и move().  
Классы-наследники должны будут переопределить их у себя и реализовать необходимую функциональность (пока можешь оставить их пустыми, просто чтобы код компилировался).  
Объяви эти методы, но сделай их абстрактными.  
Также сделай абстрактным сам класс BaseObject.

А еще нам нужно будет определять попал шарик в кирпич или в подставку.  
**Это будем делать так:**  
В этом же классе, создадим специальный метод: boolean isIntersec(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  
Если 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. Класс BaseObject должен быть абстрактным.  
2. В классе BaseObject должен быть объявлен абстрактный метод draw() c одним параметром типа Canvas.  
3. В классе BaseObject должен быть объявлен абстрактный метод move() без параметров.  
4. В классе BaseObject должен быть реализован метод isIntersec() с одним параметром типа BaseObject, возвращающий true в случае, если текущий объект пересекается с анализируемым.

### Арканоид(9)

Теперь займемся классом **Canvas**.  
Он у нас будет содержать матрицу (***двумерный массив***), 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 два приватных поля width и height.  
б) Добавить в класс приватное поле matrix (char[][]).  
в) Добавить конструктор с двумя параметрами типа int (width и height), поле matrix должно быть инициализировано пустым массивом размерностью **[**height**+2][**width**+2]**.  
г) Добавить геттеры и сеттеры для всех приватных полей класса.

### Арканоид(10)

Что мы будем делать с **Canvas**?  
Мы будем рисовать на нем (в его матрице).  
Поэтому нам понадобятся два метода  
void setPoint(double **x**, double **y**, char **c**)  
void drawMatrix(double **x**, double **y**, int[][] **matrix**, char **c**)

Первый метод — setPoint будет «ставить точку в координатах x**,**yцветомc«.

**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 элемента **[i][j]** полученной матрицы 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 должен быть создан метод void setPoint(double x, double y, char c).  
2. В классе Canvas должен быть создан метод void drawMatrix(double x, double y, int[][] matrix, char c).  
3. Поведение метода setPoint должно соответствовать условию задачи.  
4. Поведение метода drawMatrix должно соответствовать условию задачи.

### Арканоид(11)

Еще **Canvas** понадобится два метода, напиши их.  
а) метод clear()  
Этот метод будет очищать матрицу, чтобы на ней снова можно было рисовать.  
Можешь просто заново инициализировать matrix пустым массивом правильной размерности.  
б) метод print()  
Этот метод рисует матрицу на экран.  
Тут уже ты должен сам разобраться: вывести набор символов не так уж и сложно.

Требования:  
1. В классе Canvas должен быть создан метод void clear() без параметров.  
2. В классе Canvas должен быть создан метод void print() без параметров.  
3. Метод clear должен очищать массив matrix.  
4. Метод print должен выводить на экран содержимое массива matrix.

### Арканоид(12)

Теперь напишем класс **Brick**.

Во-первых займемся конструктором и в нем точно зададим «***радиус***» всех кирпичей.  
**Пусть он выглядит так:**  
public Brick(double x, double y) {  
  super(x, y, 3);  
}

Еще нам нужно переопределить два метода move() и draw(Canvas **canvas**).  
Метод move() не делает ничего (не содержит кода), т.к. кирпич никуда не двигается.

Напиши еще метод draw(Canvas **canvas**).  
Его кодом я займусь сам.

Требования:  
1. Конструктор класса Brick должен принимать два параметра типа double и вызывать конструктор класса родителя с фиксированным третьим параметром (радиусом) равным 3.  
2. В классе Brick должен существовать метод move без параметров.  
3. В классе Brick должен существовать метод draw с одним параметром типа Сanvas.

### Арканоид(13)

Класс **Ball** уже посложнее — шарик же двигается.

Нам понадобятся приватные поля:  
а) speed (скорость шарика) типа double;  
б) direction (направление движения в градусах: от0до**360**) типа double;  
в) dx (расстояние по x, которое проходит шарик за один шаг. вычисляется на основеspeedиdirection) типа double;  
г) dy (расстояние по y, которое проходит шарик за один шаг. вычисляется на основе speed и direction) типа double;  
д) isFrozen («истина» если шарик «***заморожен***» — не двигается) типа boolean.

Не забудь добавить геттеры для всех полей этого класса, кроме isFrozen (его мы будем использовать только внутри класса **Ball**).

**А еще сделай-ка конструктор:**  
а) параметры — x, y, speed, direction;  
б) радиус (для вызова родительского класса) всегда равен 1;  
в) не забудь установить isFrozen в true — в начале игры шарик никуда не летит.

Требования:  
1. Приватное поле speed типа double должно быть создано в классе Ball.  
2. Приватное поле direction типа double должно быть создано в классе Ball.  
3. Приватное поле dx должно типа double быть создано в классе Ball.  
4. Приватное поле dy должно типа double быть создано в классе Ball.  
5. Приватное поле isFrozen типа boolean должно быть создано в классе Ball.  
6. В классе Ball должен существовать корректный публичный геттер для поля speed.  
7. В классе Ball должен существовать корректный публичный геттер для поля direction.  
8. В классе Ball должен существовать корректный публичный геттер для поля dx.  
9. В классе Ball должен существовать корректный публичный геттер для поля dy.  
10. В классе Ball должен быть создан публичный конструктор (поведение и параметры описаны в условии).

### Арканоид(14)

Чего-то не хватает в классе **Ball**:

Во-первых надо реализовать метод move(), унаследованный от **BaseObject**:  
а) x должен увеличиваться на dx каждый ход  
б) y должен увеличиваться на dy каждый ход  
если шарик «***заморожен***«, то x и y меняться не должны

Во-вторых надо реализовать метод draw(Canvas **canvas**):  
на объекте canvas необходимо вызвать метод setPoint с параметрами (x, y, ‘**O**‘)

В-третьих надо реализовать метод start()  
Именно его вызов «***размораживает***» шарик.  
Что для этого надо сделать — это ты уже сам реши.

**P.S.** Чуть не забыл! Поляx, yиradiusобъявлены в классе**BaseObject**с модификатором доступа private,  
что не очень-то удобно. Измени его наprotected, чтобы все потомки класса**BaseObject**имели контроль над своими  
координатами и размерами, без необходимости пользоваться «**чужими**»***геттерами***и***сеттерами***.

### Арканоид(15)

Не поверишь, но и это еще не все.

Во-первых нужен метод setDirection,  
который не только устанавливает значение переменной direction,  
но и вычисляет новые значения переменных dx и dy.

**Код должен выглядеть примерно так:**  
this.direction = direction;  
double angel = Math.toRadians(direction);  
dx = Math.cos(angle) \* speed;  
dy = -Math.sin(angle) \* speed.

Во-вторых шарик может удариться о стенку.  
При этом он должен от нее отскочить.

**Для этого нам понадобится еще один метод:**  
void checkRebound(int **minx**, int **maxx**, int **miny**, int **maxy**)

Создай его, а кодом я займусь сам.

Требования:  
1. В классе Ball должен быть создан метод setDirection() c одним параметром типа double (реализацию смотри в условии).  
2. В классе Ball должен быть создан метод checkRebound c четырьмя параметрами типа int.

### Арканоид(16)

И наконец «***подставка***«!  
Ей понадобятся такие приватные поля:  
а) speed (скорость шарика) типа double;  
б) direction (направление движения по оси x: **1** — вправо, -1 — влево, **0** — начальное значение, стоим на месте) типа double.

**Также создай для них геттеры.**

**А еще с тебя конструктор, примерно вот такой:**  
public Stand(double x, double y) {  
super(x, y, 3);  
speed = 1;  
direction = 0;  
}

Требования:  
1. В классе Stand должно быть создано приватное поле speed типа double.  
2. В классе Stand должно быть создано приватное поле direction типа double.  
3. В классе Stand должен быть реализован публичный геттер для поля speed.  
4. В классе Stand должен быть реализован публичный геттер для поля direction.  
5. В классе Stand должен быть реализован конструктор с двумя параметрами типа double в соответствии с условием задачи.

### Арканоид(17)

**Еще подставке нужны методы:**  
а) move — см. **move** в BaseObject  
Движение доски осуществляется горизонтально, поэтому мы меняем только координату х.  
Подумай, как координата х зависит от направления (direction) и скорости (speed). Реализуй зависимость.  
б) draw — см. draw в **BaseObject**  
Его кодом я займусь сам.  
в) moveLeft() — задает постоянное движение «***подставки***» **влево**  
Просто присвой правильное значение переменной **direction**.  
г) moveRight() — задает постоянное движение «***подставки***» **вправо**  
Просто присвой правильное значение переменной direction.

### Арканоид(18)

Теперь вернемся к классу **Arkanoid**.

Реализуй методы:  
а) move()  
В этом методе нужно двигать все движимые объекты (stand**,** ball).  
б) draw(Canvas **canvas**)  
В этом методе надо вызвать метод draw всех существующих объектов, которые его имеют.

Требования:  
1. В классе Arkanoid должен быть реализован метод move() без параметров, вызывающий методы move() у шарика и подставки.  
2. В классе Arkanoid должен быть реализован метод draw(Canvas canvas) без параметров, вызывающий методы draw() у шарика, подставки и всех кирпичей.

### Арканоид(19)

Осталось совсем чуть-чуть.

В классе **Arkanoid** создай поле приватное isGameOver типа boolean.

Реализуй метод checkBricksBump.  
В этом методе надо проверить — не столкнулся ли шарик с каким-нибудь из «***кирпичей***«.  
Для проверки столкновения используй метод isIntersec.  
**Если шарик все-таки попал в кирпич, то:**  
а) шарик ***отлетает*** в случайном направлении:  
double angle = Math.random() \* 360;  
ball.setDirection(angle);  
б) кирпич ***умирает*** — надо удалить его из списка всех кирпичей.

Реализуй метод checkStandBump.  
В этом методе надо проверить — не ударился ли шарик о подставку.  
Для проверки столкновения используй метод isIntersec.  
Если шарик все-таки ударился о подставку, то:  
шарик отлетает в случайным направлении вверх:  
double angle = 90 + 20 \* (Math.random() — 0.5);  
ball.setDirection(angel);

Реализуй метод checkEndGame.  
Если координата y шарика больше чем высота поля игры (height), значит шарик улетел вниз за границу экрана.  
В этом случае надо переменную isGameOver установить в **true**.

Требования:  
1. В классе Arkanoid должен быть реализован метод checkBricksBump() в соответствии с условием задачи.  
2. В классе Arkanoid должен быть реализован метод checkStandBump() в соответствии с условием задачи.  
3. В классе Arkanoid должен быть реализован метод checkEndGame() в соответствии с условием задачи.  
4. В классе Arkanoid должно быть создано приватное поле isGameOver типа boolean.

### Арканоид(20)

**Отличная работа!** Я добавил пару методов, а также класс **KeyboardObserver**.

Немного отдохни перед новым уровнем и поиграй.

**P.S.** Только не забудь отрегулировать высоту консоли.

Требования:  
1. Расслабься и получай удовольствие.