目录

Day12. Java

1 回顾

- 面向对象
  - 封装、继承、多态
- 封装
  - 类、对象、引用
  - 构造方法
  - this
  - 重载
  - 隐藏 private
- 继承
  - 作用：代码重用、复用
  - 子类对象
    - 父类对象和子类对象绑定，整体作为一个对象
    - 调用成员，先找子类，再找父类
  - 重写
    - 继承的方法，重新定义，重新编写，改写
  - 默认 super()
  - 手动 super(参数)
- 多态
  - 作用：一致的类型
  - 向上转型、向下转型
  - instanceof
    运行期类型识别
- 抽象类，半成品
- final
  - 常量，方法，类
- static
  - 属于类
  - static {...}
  - 使用场景：

- ◆ 共享的数据、工具方法
- 访问控制符
  - public,protected,[default],private
- 对象创建过程（10步）
- 接口
  - 作用：结构设计工具，解耦合、隔离实现
  - interface
  - implements
- 内部类
  - 非静态内部类
    - ◆ 封装一个复杂对象的局部数据，或局部逻辑
  - 静态内部类
  - 局部内部类
  - 匿名内部类

查找子串

查找所有子串出现的位置

用ArrayList存放找到的下标位置

```
package day1009;

import java.util.ArrayList;
import java.util.Arrays;
import java.util.Scanner;

public class Test1 {
   public static void main(String[] args) {
      System.out.println("输入字符串：");
      String s = new Scanner(System.in).nextLine();
      System.out.println("查找的目标子串：");
      String t = new Scanner(System.in).nextLine();

      ArrayList<Integer> a = find(s, t);
      if(a.size() == 0) {
         System.out.println("找不到子串");
      } else {
         System.out.println(a);
      }
   }

   private static ArrayList<Integer> find(String s, String t) {
      /*
       * s  "aaaaaaaaaaa"
       *     |
       *     index
       *
       * t  "a"
       *
       *    [0,5,9                    ]
```

```
         *            i
         */
        //新建足够长的数组

        ArrayList<Integer> list = new ArrayList<>();
        //int[] a = new int[s.length()];
        int index = 0;
        //int i = 0;
        while(true) {
            //查找
            index = s.indexOf(t, index);
            if(index == -1) {
                break;
            }
            list.add(index);
            index++;
        }
        return list;
    }
}
```

2 HashMap

- 哈希表、散列表
- 存储"键值对"数据

Key	Value
9527	唐伯虎
9528	华夫人
9529	祝枝山
9530	旺财
9531	小强

- 键:
  - 不重复
  - 无序
  - 建必须重写 equals() 和 hashCode()
    - equals() 相等，hashCode()必须相同
    - equals() 不相等，hashCode()尽量不相同
- 作用:
  - 快速查找数据

- 方法
  - put(key, value)
    添加键值对数据

  - get(key)
    用键，提取对应的值，

键不存在，返回 null

- remove(key)
  移除键值对
  返回被移除的值

- size()

- keySet()
  把所有的键，单独创建一个 Set 集合，
  对Set中的键，可以迭代遍历

HashMap

项目：day1201_HashMap
类：day1201.Test1

```
package day1201;

import java.util.HashMap;

public class Test1 {
   public static void main(String[] args) {
      HashMap<Integer, String> map = new HashMap<>();
      map.put(9527, "唐伯虎");
      map.put(9528, "华夫人");
      map.put(9529, "旺财");
      map.put(9530, "小强");
      map.put(9531, "石榴姐");
      map.put(9532, "秋香");
      map.put(9532, "如花");
      map.put(9533, null);
      map.put(null, "---");
      System.out.println(map.size());
      System.out.println(map);
      System.out.println(map.get(9527));
      System.out.println(map.get(9999));
      System.out.println(map.remove(9530));
      System.out.println(map);
   }
}
```

Test2

```
package day1201;

import java.util.HashMap;
import java.util.Scanner;
```

```
public class Test2 {
   public static void main(String[] args) {
      /*
       * - 统计字符串中每个字符出现的次数
       *
       *   "abaccdcce"
       *        i
       *
       *   key  value
       *   a    2
       *   b    1
       *   c    2
       */
      System.out.println("输入: ");
      String s = new Scanner(System.in).nextLine();

      HashMap<Character, Integer> map = new HashMap<>();
      /*
       * *) i循环, 遍历字符串 s
       *     *) 取出i位置字符存到变量 c
       *     *) 从map提取c对应的计数值
       *         - 存到Ingeger变量count
       *     *) 如果count是null
       *          *)放入 c 和 1
       *     *) 否则, 放入 c 和 count+1
       *
       * *) 循环结束, 打印map
       */
      for(int i=0;i<s.length();i++) {
         char c = s.charAt(i);
         Integer count = map.get(c);
         if(count == null) {
            map.put(c, 1);
         } else {
            map.put(c, count+1);
         }
      }

      System.out.println(map);
   }
}
```

2.1 HashMap存入数据运算过程:

- key.hashCode(), 获得键的哈希值
- 用哈希值, 计算下标i

- 新建Entry类型对象，封装键和值
- Entry对象，放入 i 位置
  - 空位置，直接放入
  - 有数据，依次equals()比较是否相等
    - 找到相等的，覆盖值
    - 没有相等的，链表连接在一起
  - 负载率、加载因子到 0.75
    - 新建翻倍长度的新数组
    - 所有数据，重新执行哈希运算，放入新数组
  - jdk1.8
    - 链表长度到8，转成红黑树
    - 数据减少到6，转回成链表

- hashCode() 方法
  - 从Object继承的方法
  - 默认实现：
    用对象的内存地址，作为哈希值

哈希算法

坐标点，对应销售额

Point

```
package day1201;

public class Point {
   private int x;
   private int y;

   public Point() {
      super();
   }
   public Point(int x, int y) {
      super();
      this.x = x;
      this.y = y;
   }
   public int getX() {
      return x;
   }
   public void setX(int x) {
      this.x = x;
   }
   public int getY() {
      return y;
   }
   public void setY(int y) {
```

```
        this.y = y;
    }

    @Override
    public String toString() {
        return "("+x+", "+y+")";
    }

    @Override
    public int hashCode() {
        /*
         * *) 用属性数据，计算产生哈希值
         * *) 相同属性，哈希值必须相同
         * *) 不同属性，哈希值尽量不相同
         *
         * *) 数学家发明了一种算法
         * *) 是一种惯用的，有效算法
         */
        int p = 31;
        int r = 1;
        r = r*p + x;
        r = r*p + y;
        return r;
    }

    @Override
    public boolean equals(Object obj) {
        if(obj == null) return false;
        if(obj == this) return true;
        if(! (obj instanceof Point)) return false;

        Point p = (Point) obj;
        return x == p.x && y == p.y;
    }
}
```

Test3

```
package day1201;

import java.util.HashMap;

public class Test3 {
    public static void main(String[] args) {
        Point a = new Point(1, 3);
        Point b = new Point(1, 3);
        //两个对象哈希值相同,
        //才能保证计算出相同的位置
        System.out.println(a.hashCode());
        System.out.println(b.hashCode());

        //即使计算出相同位置
```

```
        //equals()必须相等，才能覆盖
        System.out.println(a.equals(b));

        HashMap<Point, String> map = new HashMap<>();
        map.put(a, "2.9亿");
        map.put(b, "3.1亿");
        System.out.println(map);
    }
}
```

3 HashSet

- Set
  ```
  |- HashSet
  |- TreeSet
  ```
- Set中的数据，不重复
- HashSet
  - 不重复
  - 无序
- HashSet内部封装了一个HashMap，
  使用HashMap的键这一列来存放数据
- 方法
  - add(数据)
  - remove(数据)
  - size()
  - iterator()

4 Iterator 接口

- 迭代器的父接口
- 迭代遍历期间，不允许用集合直接增删数据
- 方法
  - next()
    获取下一项
  - hasNext()

判断是否还有下一项

- remove()
  移除刚刚取出的数据

5 Collectons 工具类

- Collections.addAll(集合,值1,值2,值3...)
  向集合添加多个数据
- Collections.sort(List)
  排序
- Collections.binarySearch(List, 目标值)
  二分法查找
- Collections.swap(List, i, j)
  交换
- Collections.max()
- Collections.min()

Collections

项目：day1202_Collections
类：day1202.Test1

```
package day1202;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.List;

public class Test1 {
   public static void main(String[] args) {
       /*
        * "1"
        * "10"
        * "11"
        * "12"
        * "13"
        * "2"
        * "20"
        * "21"
        * "22"
        * "3"
        * "4"
        */
      List<String> list = new ArrayList<>();
```

```
        Collections.addAll(
         list,
         "1","21","30","10",
         "20","2","11","3");

        Collections.sort(list);
        System.out.println(list);
        System.out.println("----------------------");

        // Collections.sort(list, 比较器对象);
        Collections.sort(list, new Comparator<String>() {
           /*
            * 对参数o1和o2比较大小
            * o1大，正数
            * o1小，负数
            * 相同，0
            */
           @Override
           public int compare(String o1, String o2) {
              int a = Integer.parseInt(o1);
              int b = Integer.parseInt(o2);
              //return a-b;
              if(a > b) {
                  return 1;
              } else if(a < b) {
                  return -1;
              } else {
                  return 0;
              }
           }
        });
        System.out.println(list);


    }
}
```

6 for-each循环

- 对数组遍历，和集合迭代器遍历的语法简化
- 数组
  - ```
    for(int i=0;i<a.length;i++) {
        String s = a[i];
        //处理s
    }
    ```
  - for-each简化格式：
    ```

```
for(String s : a){
   //处理s
}
```

- 集合
  - ```
    for(Iterator<String> it = list.iterator(); it.hasNext(); ){
       String s = it.next();
       //处理s
    }
    ```
  - for-each简化语法：

    ```
    for(String s : list) {
       //处理s
    }
    ```

7 飞机大战-子弹碰撞

day1106_飞机大战 复制 day1203_飞机大战

Enemy

```
package day0804;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.util.Random;

public abstract class Enemy {
   //初始值，要在子类构造方法中赋值
   BufferedImage[] imgs;
   BufferedImage img;
   int x;
   int y;
   int r;//半径

   public void paint(Graphics g) {
      g.drawImage(img,x,y,null);
   }

   public abstract void step();
```
```

```
    public int rndX() {
        // [0, 400-图片宽度)
        return new Random().nextInt(400-img.getWidth());
    }

    public boolean isOut() {
        return y>654 || x<-img.getWidth() || x>400;
    }

    public boolean hitBullet(Bullet b) {
        //敌人的中心点坐标
        int dx = x + img.getWidth()/2;
        int dy = y + img.getHeight()/2;
        //dx,dy, 和子弹的x,y的距离
        int xx = dx-b.x;
        int yy = dy-b.y;
        double d = Math.sqrt(xx*xx + yy*yy);

        return d <= r;
    }
}
```

Ariplane

```
package day0804;

public class Airplane extends Enemy {

    public Airplane() {
        imgs = Main.airplane;
        img = imgs[0];
        x = rndX();
        y = -img.getHeight();
        r = 18;
    }

    @Override
    public void step() {
        y += 4;
    }

}
```

Bigplane

```
package day0804;

public class BigPlane extends Enemy {

    public BigPlane() {
        imgs = Main.bigPlane;
        img = imgs[0];
```

```
        x = rndX();
        y = -img.getHeight();
        r = 35;
    }

    @Override
    public void step() {
        y += 2;
    }

}
```

Bee

```
package day0804;

public class Bee extends Enemy {
    int dx;

    public Bee() {
        imgs = Main.bee;
        img = imgs[0];
        x = rndX();
        y = -img.getHeight();
        dx = (Math.random()<0.5 ? -2 : 2);
        r = 25;
    }

    @Override
    public void step() {
        y += 3;
        x += dx;
    }

}
```

GamePanel

```
package day0804;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;

import javax.swing.JPanel;

public class GamePanel extends JPanel {
    Background bg = new Background();
```

```
    Hero hero = new Hero();
    ArrayList<Enemy> enemys = new ArrayList<>();
    ArrayList<Bullet> bullets = new ArrayList<>();
    //帧计数
    static int count;

    public GamePanel() {
        //设置面板的期望大小
        setPreferredSize(new Dimension(400, 654));
    }

    /*
     * 固定的绘图方法
     * 由系统自动调用
     */
    @Override
    public void paint(Graphics g) {
        bg.paint(g);
        paintEnemys(g);//绘制所有敌人
        paintBullets(g);//绘制所有子弹
        hero.paint(g);
    }

    private void paintBullets(Graphics g) {
        Iterator<Bullet> it = bullets.iterator();
        while(it.hasNext()) {
            Bullet b = it.next();
            b.paint(g);
        }
    }

    private void paintEnemys(Graphics g) {
        Iterator<Enemy> it = enemys.iterator();
        while(it.hasNext()) {
            Enemy e = it.next();
            e.paint(g);
        }
    }

    //动起来方法
    public void action() {
        //设置鼠标监听器
        addMouseMotionListener(new MouseAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                hero.setTarget(e.getX(), e.getY());
            }
        });


        //画面一帧一帧的循环播放
        while(true) {
            count++;//帧计数增加
```

```
            bg.step();//背景移动
            enemysIn();//敌人出现
            enemysStep();//所有敌人移动一帧
            bulletsIn();//子弹出现
            bulletsStep();//所有子弹移动一帧
            hero.step();

            //敌人和子弹碰撞
            hitBullet();

            //通知底层系统，重绘画面
            //系统受到通知后，会自动调用 paint() 方法
            repaint();
            //暂停 1/60 秒 (60 fps)
            try {
                Thread.sleep(1000/60);
            } catch (InterruptedException e) {
            }
        }
    }

    private void hitBullet() {
        //遍历敌人
        for (Enemy e : enemys) {
            //判断所有子弹，是否碰撞敌人e
            hitBullet(e);
        }
    }

    private void hitBullet(Enemy e) {
        //遍历子弹
        for (Iterator it = bullets.iterator(); it.hasNext();) {
            Bullet b = (Bullet) it.next();
            //敌人e是否碰撞子弹b
            if(e.hitBullet(b)) {
                it.remove(); //子弹消失
                // TODO: 敌人生命值减少
                // TODO: 英雄得分增加
            }
        }
    }



    private void bulletsIn() {
        //每15帧发射一次子弹
        if(count%15 == 0) {
            ArrayList<Bullet> list = hero.shoot();
            bullets.addAll(list);
        }
    }
```

```
    private void bulletsStep() {
        Iterator<Bullet> it = bullets.iterator();
        while(it.hasNext()) {
            Bullet b = it.next();
            b.step();
            if(b.isOut()) {
                it.remove();
            }
        }
    }


    private void enemysIn() {
        //每30帧出现
        if(count%30 == 0) {
            double r = Math.random();
            if(r<0.6) {
                enemys.add(new Airplane());
            } else if(r<0.9) {
                enemys.add(new BigPlane());
            } else {
                enemys.add(new Bee());
            }
        }
    }

    private void enemysStep() {
        Iterator<Enemy> it = enemys.iterator();
        while(it.hasNext()) {
            Enemy e = it.next();
            e.step();
            if(e.isOut()) {
                /*
                 * 再迭代遍历期间,
                 * 不允许使用集合来增删数据
                 * 如果要删除数据，必须使用迭代器的删除方法
                 *
                 */
                //enemys.remove(e);
                it.remove();//删除刚刚取出的数据
            }
        }
    }
}
```

8 飞机大战-敌人爆炸

- 敌人生命值
  - Enemy 添加 life变量
    - Airplane, life=2
    - BigPlane, life=5
    - Bee, life=1

- 敌人状态
  - 正常 STAT_NORMAL=0
  - 爆炸 STAT_BOOM=1
  - 死亡 STAT_DEAD=2
  - Enemy中添加状态变量 stat

- 判断敌人状态的方法
  - Enemy中添加方法
    - isNormal()
    - isBoom()
    - isDead()

- 敌人状态切换
  - 碰撞子弹，生命值变成0，切换到BOOM
  - 爆炸动画结束，切换到 DEAD

- 爆炸动画
  - 在Enemy中添加 boom() 方法
    - 每6帧，切换一张图片
    - 最后一张显示6帧之后，切换到 DEAD
  - 在while(true)死循环，
    - 如果敌人状态是NORMAL，e.step()
    - 如果敌人状态是BOOM，e.boom()
    - 如果敌人状态是DEAD，从集合移除

day1203_飞机大战
复制
day1204_飞机大战

Enemy

```
package day0804;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.util.Random;

public abstract class Enemy {
   static final int STAT_NORMAL   =0;
   static final int STAT_BOOM     =1;
   static final int STAT_DEAD     =2;
```

```
    //初始值，要在子类构造方法中赋值
    BufferedImage[] imgs;
    BufferedImage img;
    int x;
    int y;
    int r;//半径
    int life;//生命值
    int stat = STAT_NORMAL;//状态

    int index;//爆炸时，下标递增，切换访问数组图片

    public void paint(Graphics g) {
        g.drawImage(img,x,y,null);
    }

    public abstract void step();

    public int rndX() {
        // [0, 400-图片宽度)
        return new Random().nextInt(400-img.getWidth());
    }

    public boolean isOut() {
        return y>654 || x<-img.getWidth() || x>400;
    }

    public boolean hitBullet(Bullet b) {
        //非正常状态，不检测碰撞
        if(stat != STAT_NORMAL) {
            return false;
        }
        //敌人的中心点坐标
        int dx = x + img.getWidth()/2;
        int dy = y + img.getHeight()/2;
        //dx,dy，和子弹的x,y的距离
        int xx = dx-b.x;
        int yy = dy-b.y;
        double d = Math.sqrt(xx*xx + yy*yy);

        boolean peng = d <= r;
        if(peng) {
            life--;
            if(life == 0) {
                //切换到BOOM状态
                stat = STAT_BOOM;
            }
        }

        return peng;
    }

    public boolean isNormal() {
        return stat == STAT_NORMAL;
```

```
    }
    public boolean isBoom() {
        return stat == STAT_BOOM;
    }
    public boolean isDead() {
        return stat == STAT_DEAD;
    }

    public void boom() {
        //每6帧，换一张爆炸图
        if(GamePanel.count%6 == 0) {
            if(index == 5) {
                //状态切换到 DEAD
                stat = STAT_DEAD;
                return;
            }
            img = imgs[index++];
        }
    }
}
```

Ariplane

```
package day0804;

public class Airplane extends Enemy {

    public Airplane() {
        imgs = Main.airplane;
        img = imgs[0];
        x = rndX();
        y = -img.getHeight();
        r = 18;
        life = 2;
    }

    @Override
    public void step() {
        y += 4;
    }

}
```

BigPlane

```
package day0804;

public class BigPlane extends Enemy {

    public BigPlane() {
        imgs = Main.bigPlane;
        img = imgs[0];
```

```
        x = rndX();
        y = -img.getHeight();
        r = 35;
        life = 5;
    }

    @Override
    public void step() {
        y += 2;
    }

}
```

Bee

```
package day0804;

public class Bee extends Enemy {
    int dx;

    public Bee() {
        imgs = Main.bee;
        img = imgs[0];
        x = rndX();
        y = -img.getHeight();
        dx = (Math.random()<0.5 ? -2 : 2);
        r = 25;
        life = 1;
    }

    @Override
    public void step() {
        y += 3;
        x += dx;
    }

}
```

GamePanel

```
package day0804;

import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.event.MouseAdapter;
import java.awt.event.MouseEvent;
import java.util.ArrayList;
import java.util.Iterator;

import javax.swing.JPanel;

public class GamePanel extends JPanel {
```

```
    Background bg = new Background();
    Hero hero = new Hero();
    ArrayList<Enemy> enemys = new ArrayList<>();
    ArrayList<Bullet> bullets = new ArrayList<>();
    //帧计数
    static int count;

    public GamePanel() {
        //设置面板的期望大小
        setPreferredSize(new Dimension(400, 654));
    }

    /*
     * 固定的绘图方法
     * 由系统自动调用
     */
    @Override
    public void paint(Graphics g) {
        bg.paint(g);
        paintEnemys(g);//绘制所有敌人
        paintBullets(g);//绘制所有子弹
        hero.paint(g);
    }

    private void paintBullets(Graphics g) {
        Iterator<Bullet> it = bullets.iterator();
        while(it.hasNext()) {
            Bullet b = it.next();
            b.paint(g);
        }
    }

    private void paintEnemys(Graphics g) {
        Iterator<Enemy> it = enemys.iterator();
        while(it.hasNext()) {
            Enemy e = it.next();
            e.paint(g);
        }
    }

    //动起来方法
    public void action() {
        //设置鼠标监听器
        addMouseMotionListener(new MouseAdapter() {
            @Override
            public void mouseMoved(MouseEvent e) {
                hero.setTarget(e.getX(), e.getY());
            }
        });


        //画面一帧一帧的循环播放
        while(true) {
            count++;//帧计数增加
```

```
            bg.step();//背景移动
            enemysIn();//敌人出现
            enemysStep();//所有敌人移动一帧
            bulletsIn();//子弹出现
            bulletsStep();//所有子弹移动一帧
            hero.step();

            //敌人和子弹碰撞
            hitBullet();

            //通知底层系统，重绘画面
            //系统受到通知后，会自动调用 paint() 方法
            repaint();
            //暂停 1/60 秒 (60 fps)
            try {
                Thread.sleep(1000/60);
            } catch (InterruptedException e) {
            }
        }
    }

    private void hitBullet() {
        //遍历敌人
        for (Enemy e : enemys) {
            //判断所有子弹，是否碰撞敌人e
            hitBullet(e);
        }
    }

    private void hitBullet(Enemy e) {
        //遍历子弹
        for (Iterator it = bullets.iterator(); it.hasNext();) {
            Bullet b = (Bullet) it.next();
            //敌人e是否碰撞子弹b
            if(e.hitBullet(b)) {
                it.remove(); //子弹消失
                // TODO: 敌人生命值减少
                // TODO: 英雄得分增加
            }
        }
    }



    private void bulletsIn() {
        //每15帧发射一次子弹
        if(count%15 == 0) {
            ArrayList<Bullet> list = hero.shoot();
            bullets.addAll(list);
        }
    }
```

```
    private void bulletsStep() {
        Iterator<Bullet> it = bullets.iterator();
        while(it.hasNext()) {
            Bullet b = it.next();
            b.step();
            if(b.isOut()) {
                it.remove();
            }
        }
    }


    private void enemysIn() {
        //每30帧出现
        if(count%30 == 0) {
            double r = Math.random();
            if(r<0.6) {
                enemys.add(new Airplane());
            } else if(r<0.9) {
                enemys.add(new BigPlane());
            } else {
                enemys.add(new Bee());
            }
        }
    }

    private void enemysStep() {
        Iterator<Enemy> it = enemys.iterator();
        while(it.hasNext()) {
            Enemy e = it.next();
            if(e.isNormal()) {
                e.step();
            } else if(e.isBoom()) {
                e.boom();
                continue;
            } else {
                it.remove();
                continue;
            }

            if(e.isOut()) {
                /*
                 * 再迭代遍历期间,
                 * 不允许使用集合来增删数据
                 * 如果要删除数据，必须使用迭代器的删除方法
                 *
                 */
                //enemys.remove(e);
                it.remove();//删除刚刚取出的数据
            }

        }
    }
}
```

9 作业

- 用 HashMap 保存学生对象对应成绩

  HashMap<Student, Integer>

  重复放入同一个学生，替换旧的成绩
  再学生类中，重写 hashCode() 和 equals()

  ```
  ... + id
  ... + age
  ... + name.hashCode()
  ... + gender.hashCode()
  ```

  模仿坐标对象对应销售额

- 背诵 HashMap 的哈希算法