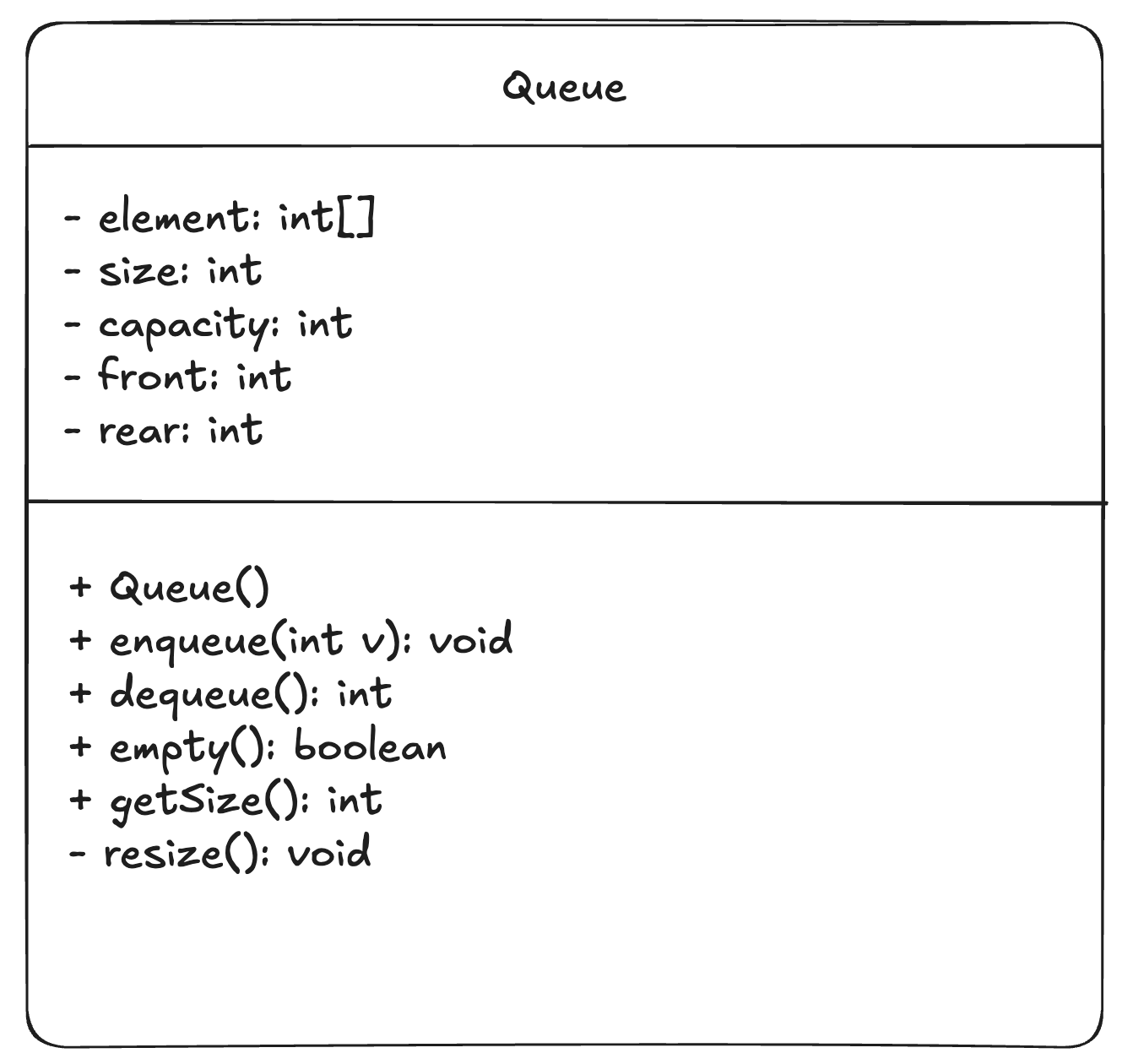
**实验三**

**1.题目：**

|  |
| --- |
| \*10.10  (Queue 类)10.6 节给出了一个 Stack类。设计一个名为 Queue 的类用于存储整数。像栈一样,队列保存元素。在栈中，元素以“后进先出”的方式获取。在队列中，元素以“先进先出”的方式获取。该类包含:  。一个名为 element 的 int[]类型的数据域，保存队列中的 int 值。  。一个名为 size 的数据域，保存队列中的元素个数。  。一个构造方法，以默认容量为8来创建一个 Queue 对象。  。方法 enqueue(int v)，用于将v加人队列中。  。方法 dequeue()，用于从队列中移除元素并返回该元素。  。方法 empty()，如果队列为空，该方法返回 true。  。方法 getSize()，返回队列的大小。  画出该类的 UML 图并实现这个类，初始数组的大小为 8。一旦元素个数超过了数组大小，数组大小将会翻倍。如果一个元素从数组的开始部分移除，你需要将数组中的所有元素往左边移动一个位置。编写一个测试程序，添加从1到20的20个数字到队列中，然后将这些数字移除并显示它们。 |

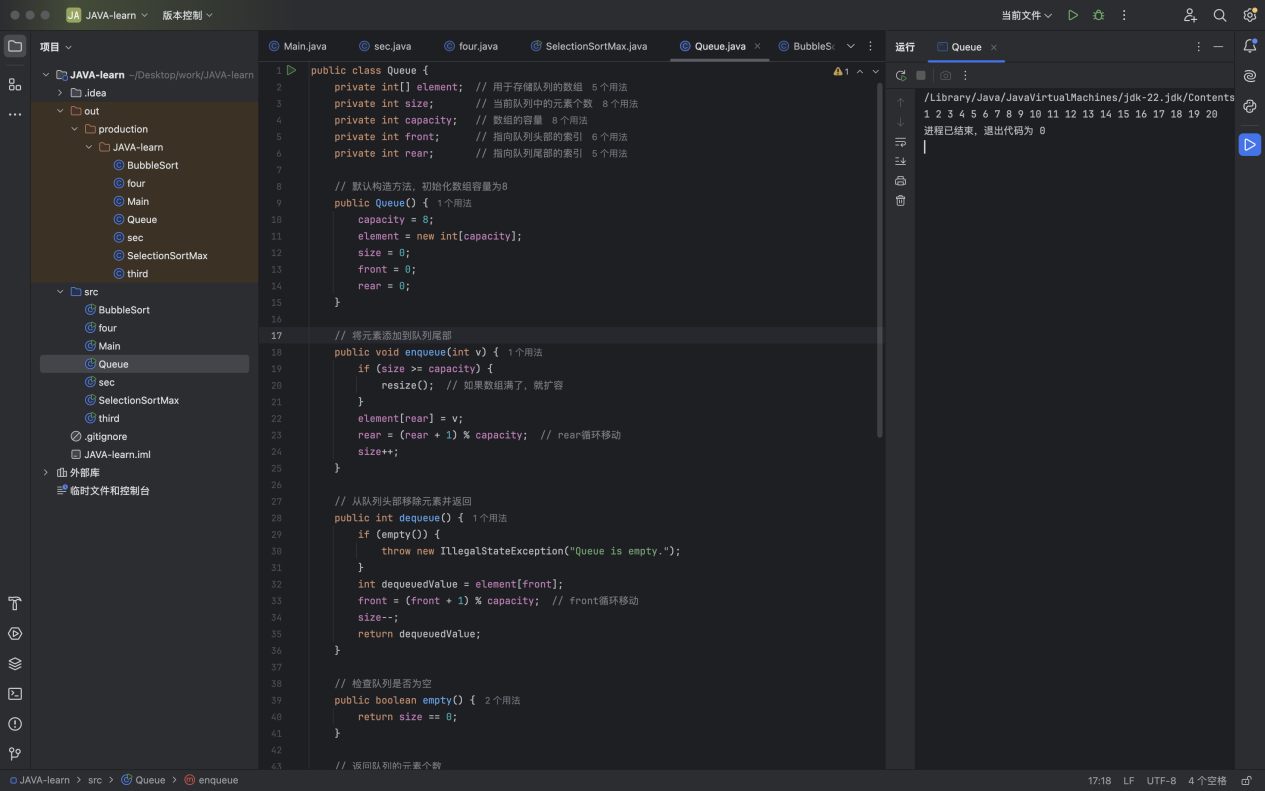
**UML导图：**

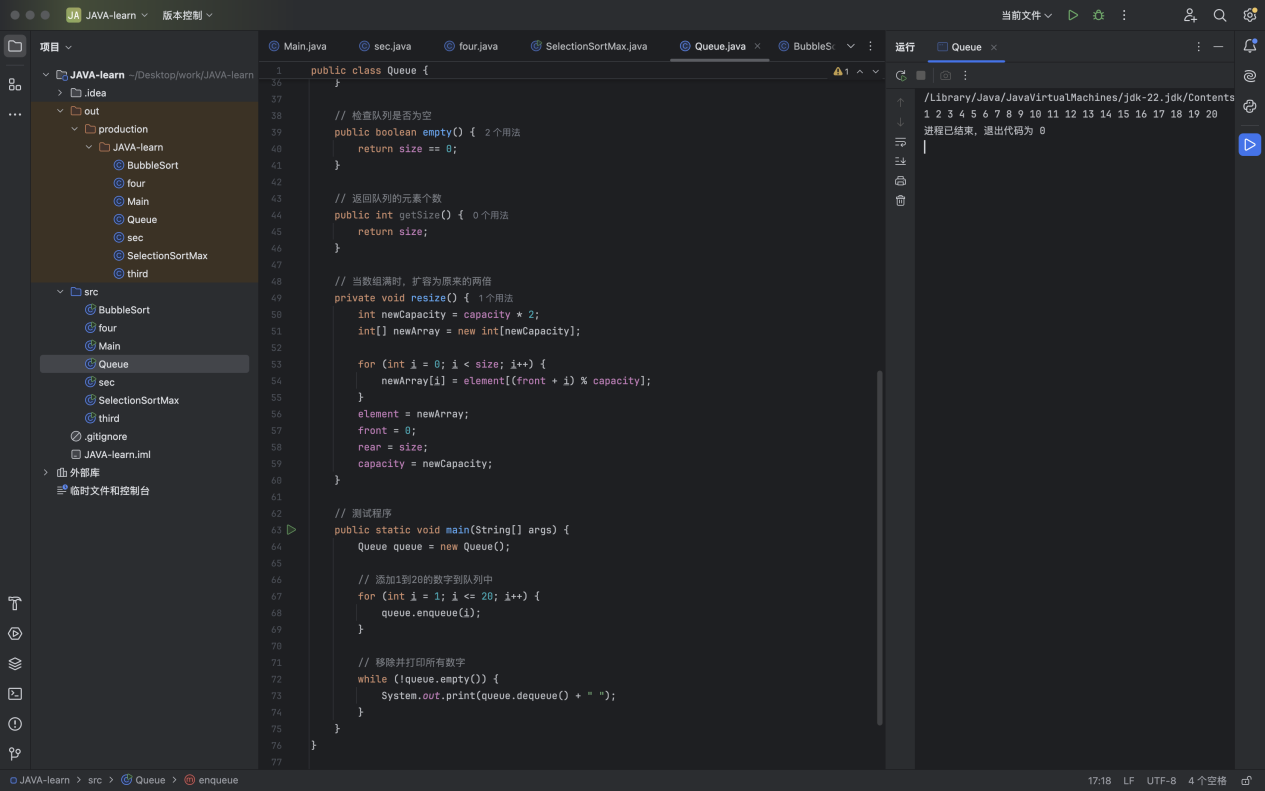


**完整代码：**

|  |
| --- |
| JavaScript public class Queue {  private int[] element; // 用于存储队列的数组  private int size; // 当前队列中的元素个数  private int capacity; // 数组的容量  private int front; // 指向队列头部的索引  private int rear; // 指向队列尾部的索引   // 默认构造方法，初始化数组容量为8  public Queue() {  capacity = 8;  element = new int[capacity];  size = 0;  front = 0;  rear = 0;  }   // 将元素添加到队列尾部  public void enqueue(int v) {  if (size >= capacity) {  resize(); // 如果数组满了，就扩容  }  element[rear] = v;  rear = (rear + 1) % capacity; // rear循环移动  size++;  }   // 从队列头部移除元素并返回  public int dequeue() {  if (empty()) {  throw new IllegalStateException("Queue is empty.");  }  int dequeuedValue = element[front];  front = (front + 1) % capacity; // front循环移动  size--;  return dequeuedValue;  }   // 检查队列是否为空  public boolean empty() {  return size == 0;  }   // 返回队列的元素个数  public int getSize() {  return size;  }   // 当数组满时，扩容为原来的两倍  private void resize() {  int newCapacity = capacity \* 2;  int[] newArray = new int[newCapacity];   for (int i = 0; i < size; i++) {  newArray[i] = element[(front + i) % capacity];  }  element = newArray;  front = 0;  rear = size;  capacity = newCapacity;  }   // 测试程序  public static void main(String[] args) {  Queue queue = new Queue();   // 添加1到20的数字到队列中  for (int i = 1; i <= 20; i++) {  queue.enqueue(i);  }   // 移除并打印所有数字  while (!queue.empty()) {  System.*out*.print(queue.dequeue() + " ");  }  } } |

**代码运行截图：**

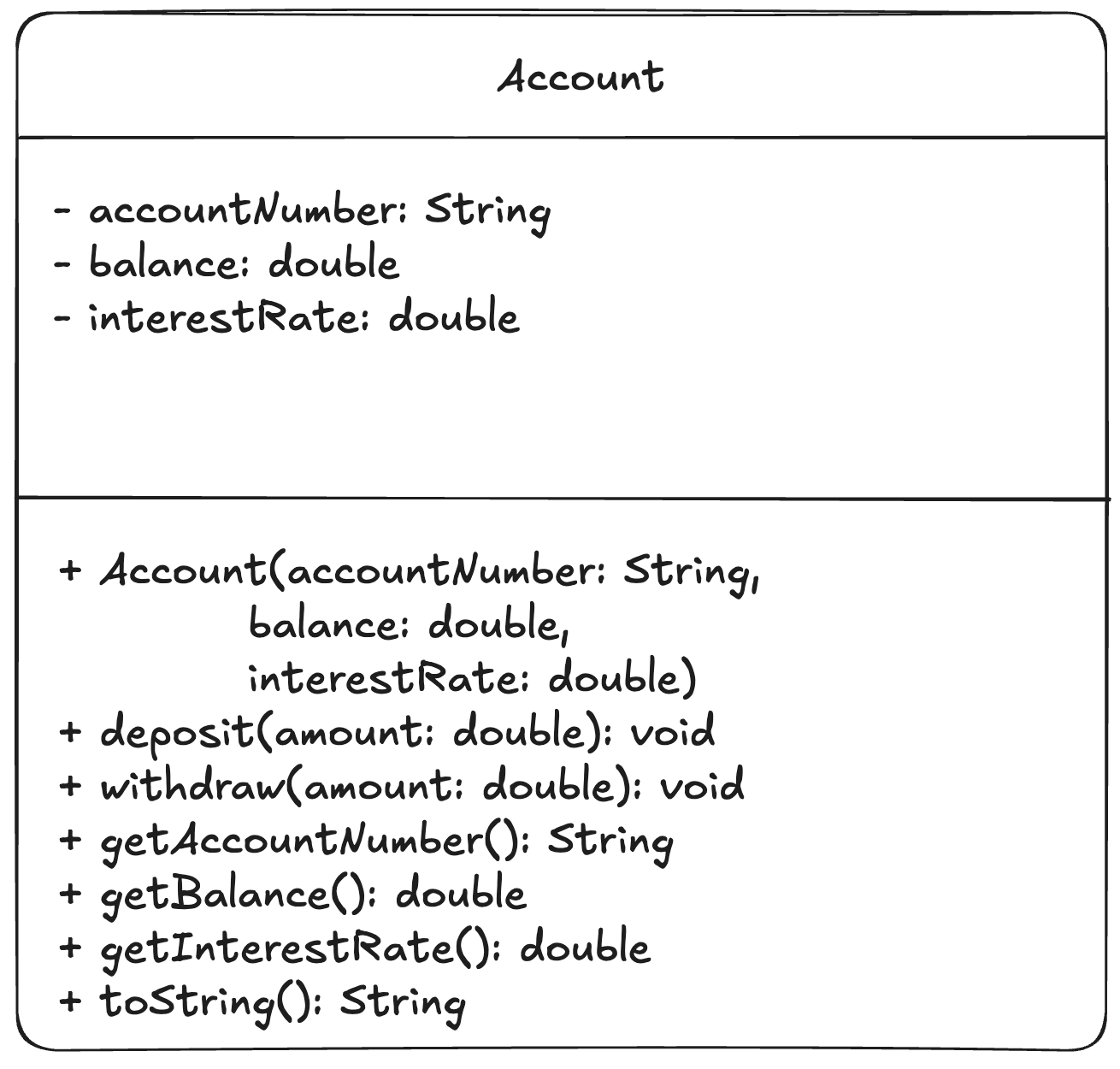




**2.题目：**

|  |
| --- |
| •2、定义Accout类，有账号、余额、利率等属性，有存款取款等行为。  •画出UML类图，实现并测试这些类，并调用它们的toString（）方法。 |

**UML导图：**



**完整代码：**

|  |
| --- |
| JavaScript public class Account {  private String accountNumber; // 账户号码  private double balance; // 账户余额  private double interestRate; // 利率   // 构造方法，初始化账户信息  public Account(String accountNumber, double balance, double interestRate) {  this.accountNumber = accountNumber;  this.balance = balance;  this.interestRate = interestRate;  }   // 存款方法，将金额存入账户  public void deposit(double amount) {  if (amount > 0) {  balance += amount;  } else {  System.*out*.println("存款金额必须大于0");  }  }   // 取款方法，从账户中取款  public void withdraw(double amount) {  if (amount > 0 && amount <= balance) {  balance -= amount;  } else {  System.*out*.println("取款金额无效或余额不足");  }  }   // 获取账户号码  public String getAccountNumber() {  return accountNumber;  }   // 获取账户余额  public double getBalance() {  return balance;  }   // 获取利率  public double getInterestRate() {  return interestRate;  }   // 返回账户信息的字符串表示  @Override  public String toString() {  return "Account Number: " + accountNumber + "\n" +  "Balance: " + balance + "\n" +  "Interest Rate: " + interestRate + "%";  }   // 测试程序  public static void main(String[] args) {  // 创建一个对象，初始余额为1000，利率为1.5%  Account account = new Account("123456789", 1000.0, 1.5);   // 调用deposit和withdraw方法  account.deposit(500.0);  account.withdraw(200.0);   // 打印账户信息，调用toString()方法  System.*out*.println(account.toString());  } } |

**代码运行截图**

