7.3 成员方法传参机制（非常非常重要）

在 Java 编程中，深入理解成员方法的传参机制对于编写高效、准确的代码至关重要。接下来，我们将通过具体案例详细探讨不同数据类型在方法传参时的行为和特点。

7.3.1 基本数据类型的传参机制

案例代码（MethodParameter01.java）

```
public class MethodParameter01 {
    // 编写一个main方法
    public static void main(String[] args) {
        int a = 10;
        int b = 20;
        // 创建AA对象名字obj
        AA obj = new AA();
        obj.swap(a, b); // 调用swap
        System.out.println("main方法a=" + a + " b=" + b);// a=10 b=20
    }
}

class AA {
    public void swap(int a, int b) {
        System.out.println("\na和b交换前的值\na=" + a + "\tb=" + b);// a=10 b=20
        // 完成了a和b的交换
        int tmp = a;
        a = b;
        b = tmp;
        System.out.println("\na和b交换后的值\na=" + a + "\tb=" + b);// a=20 b=10
    }
}
```

代码解读

在`main`方法中，我们定义了两个基本数据类型变量`a`和`b`，分别赋值为 10 和 20。然后创建了`AA`类的对象`obj`，并调用`obj`的`swap`方法，将`a`和`b`作为实参传递进去。在`swap`方法中，首先输出交换前`a`和`b`的值，然后通过中间变量`tmp`实现`a`和`b`值的交换，并再次输出交换后的值。然而，当我们在`main`方法中再次输出`a`和`b`的值时，会发现它们并没有发生改变，仍然是初始值 10 和 20。这是因为在 Java 中，基本数据类型传参时，传递的是值的副本。也就是说，在调用`swap`方法时，实参`a`和`b`的值被复制给了形参`a`和`b`，在`swap`方法中对形参的操作不会影响到实参。可以通过以下示意图来理解：

（此处插入一个简单的示意图，展示基本数据类型传参时，实参和形参在栈内存中的关系，以及方法调用前后值的变化情况）通过这个案例，我们清楚地看到了基本数据类型传参的机制，即方法内对形参的修改不会影响到方法外的实参。

7.3.2 引用数据类型的传参机制

案例一：数组作为参数（MethodParameter02.java 部分代码）

```
public class MethodParameter02 {
    // 编写一个main方法
    public static void main(String[] args) {
        // 测试
        B b = new B();
        int[] arr = {1, 2, 3};
        b.test100(arr);// 调用方法
        System.out.println(" main的arr数组 ");
        // 遍历数组
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
        System.out.println();
    }
}

class B {
    // B类中编写一个方法test100，
    // 可以接收一个数组，在方法中修改该数组，看看原来的数组是否变化
    public void test100(int[] arr) {
        arr[0] = 200;// 修改元素
        // 遍历数组
        System.out.println(" test100的arr数组 ");
        for (int i = 0; i < arr.length; i++) {
            System.out.print(arr[i] + "\t");
        }
        System.out.println();
    }
}
```

在这个案例中，我们在`main`方法中创建了一个数组`arr`，并将其作为实参传递给`B`类的`test100`方法。在`test100`方法中，我们修改了数组的第一个元素值为 200。当在`main`方法中再次遍历数组时，会发现数组的第一个元素已经被修改为 200。这表明，引用数据类型传参时，传递的是对象的地址。也就是说，在调用`test100`方法时，实参数组`arr`的地址被传递给了形参数组`arr`，此时形参和实参指向的是堆内存中的同一个数组对象。因此，在方法内对数组元素的修改会影响到方法外的数组。

案例二：自定义对象作为参数（MethodParameter02.java 部分代码）

```
public class MethodParameter02 {
    // 编写一个main方法
    public static void main(String[] args) {
        // 测试
        B b = new B();
        Person p = new Person();
        p.name = "jack";
        p.age = 10;
        b.test200(p);
        System.out.println("main的p.age=" + p.age);// 10
    }
}
```

```
class Person {
    String name;
    int age;
}

class B {
    public void test200(Person p) {
        p.age = 10000; // 修改对象属性
        // 思考
        p = new Person();
        p.name = "tom";
        p.age = 99;
        // 思考
        // p = null;
    }
}
```

在这个案例中，我们在`main`方法中创建了一个`Person`对象`p`，并设置了其属性。然后将`p`作为实参传递给`B`类的`test200`方法。在`test200`方法中，我们首先修改了`p`对象的`age`属性值为 10000。接着，我们创建了一个新的`Person`对象并赋值给`p`，或者将`p`赋值为`null`。当在`main`方法中输出`p.age`时，会发现如果只是修改对象属性（如`p.age = 10000;`），`main`方法中的`p`对象属性会被改变；但如果重新创建对象（如`p = new Person();`）或赋值为`null`（如`p = null;`），`main`方法中的`p`对象仍然是原来的对象，其属性保持不变。这是因为引用数据类型传参传递的是地址，当修改对象属性时，通过地址找到堆内存中的对象并修改其属性值，所以会影响到方法外的对象；而当重新创建对象或赋值为`null`时，只是改变了形参`p`的指向，实参`p`的指向并未改变，所以方法外的对象不受影响。可以通过以下示意图来理解：
（此处插入两个示意图，一个展示修改对象属性时，实参和形参指向同一对象，属性值改变的情况；另一个展示重新创建对象或赋值为`null`时，形参指向改变，实参指向不变的情况）

结论

引用类型传递的是地址（传递也是值，但是值是地址），在方法中可以通过形参影响实参。当方法内通过形参修改对象属性时，会影响到实参对象；但当在方法内重新改变形参的指向（如重新创建对象或赋值为`null`）时，不会影响到实参的指向。

7.3.3 成员方法返回类型是引用类型应用实例

案例代码（MethodExercise02.java）

```
public class MethodExercise02 {
    // 编写一个main方法
    public static void main(String[] args) {
        Person p = new Person();
        p.name = "milan";
        p.age = 100;
        // 创建tools
        MyTools tools = new MyTools();
        Person p2 = tools.copyPerson(p);
        // 到此p和p2是Person对象，但是是两个独立的对象，属性相同
        System.out.println("p的属性age=" + p.age + " 名字=" + p.name);
        System.out.println("p2的属性age=" + p2.age + " 名字=" + p2.name);
        // 这里老师提示： 可以同对象比较看看是否为同一个对象
```

```
        System.out.println(p == p2);// false
    }
}

class Person {
    String name;
    int age;
}

class MyTools {
    // 编写一个方法copyPerson，可以复制一个Person对象，返回复制的对象。克隆对象，
    // 注意要求得到新对象和原来的对象是两个独立的对象，只是他们的属性相同
    //
    // 编写方法的思路
    // 1. 方法的返回类型Person
    // 2. 方法的名字copyPerson
    // 3. 方法的形参 (Person p)
    // 4. 方法体，创建一个新对象，并复制属性，返回即可
    public Person copyPerson(Person p) {
        // 创建一个新的对象
        Person p2 = new Person();
        p2.name = p.name; // 把原来对象的名字赋给p2.name
        p2.age = p.age; // 把原来对象的年龄赋给p2.age
        return p2;
    }
}
```

代码解读

在这个案例中，我们定义了`MyTools`类，其中有一个`copyPerson`方法，该方法接收一个`Person`对象作为参数，并返回一个新的`Person`对象。在`main`方法中，我们创建了一个`Person`对象`p`，并设置了其属性。然后调用`tools.copyPerson(p)`方法，将返回的新对象赋值给`p2`。通过输出`p`和`p2`的属性以及使用`==`比较两个对象，我们可以发现`p`和`p2`是两个独立的对象，虽然它们的属性相同。这是因为在`copyPerson`方法中，我们创建了一个新的`Person`对象`p2`，并将传入对象`p`的属性复制给`p2`，最后返回`p2`。这样就实现了对`Person`对象的复制，并且返回的新对象与原来的对象相互独立，各自在堆内存中有自己的空间。这种使用引用类型作为返回值的方式，在实际编程中常用于创建对象的副本、克隆对象等场景，为数据处理和对象操作提供了更多的灵活性。

一、选择题

1. 以下关于基本数据类型传参的说法，正确的是（）
   A. 方法内对形参的修改会直接影响实参
   B. 传参时传递的是实参的地址
   C. 传参时传递的是实参值的副本
   D. 基本数据类型不能作为方法参数
2. 若有方法定义`public void change(int num) { num = num + 1; }`，在`main`方法中调用`int a = 5; change(a);`后，`a`的值为（）
   A. 5 B. 6 C. 不确定 D. 编译错误
3. 对于引用数据类型传参，下列说法错误的是（）
   A. 传递的是对象的地址
   B. 方法内对对象属性的修改会影响实参对象

C. 在方法内重新创建对象会改变实参对象的引用
D. 可以通过形参影响实参

4. 定义方法`public void modifyArray(int[] arr) { arr[1] = 100; }`，在`main`方法中调用`int[] array = {1, 2, 3}; modifyArray(array);`后，`array[1]`的值为（ ）
A. 2 B. 100 C. 3 D. 不确定
5. 若有类`class Person { int age; }`，以及方法`public void updatePerson(Person p) { p.age = 25; }`，在`main`方法中调用`Person person = new Person(); person.age = 20; updatePerson(person);`后，`person.age`的值为（ ）
A. 20 B. 25 C. 不确定 D. 编译错误
6. 方法`public void test(String str) { str = "new value"; }`，在`main`方法中调用`String s = "old value"; test(s);`后，`s`的值为（ ）
A. "new value" B. "old value" C. 不确定 D. 编译错误
7. 定义方法`public Person createNewPerson() { Person p = new Person(); p.name = "Tom"; p.age = 30; return p; }`，调用该方法后得到的返回值是（ ）
A. 一个新的`Person`对象
B. 原有的`Person`对象
C. `null`
D. 不确定
8. 以下哪种情况会改变实参对象的引用（ ）
A. 在方法内修改对象的属性值
B. 在方法内重新创建一个新对象并赋值给形参
C. 在方法内将形参赋值为`null`
D. 以上都不会
9. 方法`public void operate(int num1, int num2) { int temp = num1; num1 = num2; num2 = temp; }`，在`main`方法中调用`int a = 10, b = 20; operate(a, b);`后，`a`和`b`的值分别为（ ）
A. 10, 20 B. 20, 10 C. 10, 10 D. 20, 20
10. 若有方法`public void process(List list) { list.add("new element"); }`，在`main`方法中调用`List myList = new ArrayList(); myList.add("old element"); process(myList);`后，`myList`的大小为（ ）
A. 1 B. 2 C. 不确定 D. 编译错误

二、填空题

1. 基本数据类型传参时，传递的是__，方法内对形参的修改__（会 / 不会）影响实参。
2. 引用数据类型传参时，传递的是__，方法内对对象属性的修改__（会 / 不会）影响实参对象。
3. 若方法接收一个数组作为参数，在方法内修改数组元素，原来的数组__（会 / 不会）发生变化，因为__。
4. 对于自定义对象作为参数传递，在方法内重新创建对象并赋值给形参，实参对象__（会 / 不会）受到影响，此时形参和实参__（指向 / 不指向）同一个对象。
5. 方法返回引用类型对象时，返回的是__，可以通过返回的对象进行__。
6. 定义方法`public void modify(int[] arr) { arr[0] = 99; }`，在`main`方法中调用`int[] numbers = {1, 2, 3}; modify(numbers);`后，`numbers[0]`的值变为__。
7. 若有类`class Car { String brand; }`，方法`public void changeCar(Car car) { car.brand = "BMW"; }`，在`main`方法中调用`Car myCar = new Car(); myCar.brand = "Toyota"; changeCar(myCar);`后，`myCar.brand`的值为__。
8. 方法`public Person clonePerson(Person p) { Person newP = new Person(); newP.name = p.name; newP.age = p.age; return newP; }`，调用该方法后，新创建的`Person`对象和传入的`Person`对象是__（同一个 / 不同的）对象，它们的属性值__（相同 / 不同）。

9. 基本数据类型传参机制使得方法内的操作具有__，引用数据类型传参机制便于实现对象的__操作。
10. 当在方法内将引用类型参数赋值为`null`时，实参对象__（会 / 不会）变为`null`，因为__。

三、判断题

1. 基本数据类型和引用数据类型传参机制相同。（ ）
2. 引用数据类型传参时，在方法内对对象的任何操作都会影响实参对象。（ ）
3. 方法返回引用类型对象时，返回的是对象的副本。（ ）
4. 基本数据类型作为方法参数时，在方法内修改形参不会影响程序其他部分。（ ）
5. 对于引用数据类型传参，只要在方法内不重新创建对象，就不会改变实参对象的引用。（ ）
6. 若方法接收一个字符串作为参数，在方法内修改字符串内容，实参字符串会相应改变。（ ）
7. 定义方法`public void update(int num) { num++; }`，调用该方法后，传入的实参值会增加。（ ）
8. 引用数据类型传参时，形参和实参指向堆内存中的同一个对象。（ ）
9. 方法返回引用类型对象可以用于创建对象的副本，实现数据的独立操作。（ ）
10. 基本数据类型传参时，实参和形参在栈内存中占用不同的空间。（ ）

四、简答题

1. 简述基本数据类型传参机制的原理，并举例说明。
2. 说明引用数据类型传参机制与基本数据类型传参机制的主要区别。
3. 对于引用数据类型传参，在方法内重新创建对象和修改对象属性有何不同？对实参对象有什么影响？
4. 方法返回引用类型对象有什么作用？请举例说明应用场景。
5. 请分析以下代码中基本数据类型和引用数据类型参数在方法调用过程中的变化情况：

```
public class Test {
    public static void main(String[] args) {
        int num = 5;
        int[] arr = {1, 2, 3};
        change(num, arr);
        System.out.println("num: " + num);
        System.out.println("arr[0]: " + arr[0]);
    }
    public static void change(int n, int[] a) {
        n = n + 1;
        a[0] = a[0] + 1;
    }
}
```

1. 为什么在方法内对基本数据类型形参的修改不会影响实参，而对引用数据类型对象属性的修改会影响实参对象？
2. 若有一个方法接收一个自定义对象作为参数，如何在方法内确保不改变实参对象的引用？
3. 举例说明如何通过方法返回引用类型对象实现对象的复制，并解释复制后的对象与原对象的关系。
4. 简述在实际编程中，理解和掌握传参机制的重要性。
5. 对于引用数据类型传参，当在方法内将形参赋值为`null`时，实参对象会发生什么变化？为什么？

五、编程题

1. 编写一个方法 `swapNumbers`，接收两个整数作为参数，在方法内交换这两个整数的值。在 `main` 方法中调用该方法，并输出交换前后整数的值，观察基本数据类型传参的效果。
2. 定义一个类 `Student`，包含 `name` 和 `age` 属性。编写一个方法 `updateStudent`，接收一个 `Student` 对象作为参数，在方法内修改对象的 `age` 属性。在 `main` 方法中创建 `Student` 对象，调用 `updateStudent` 方法，并输出修改后的 `Student` 对象属性，展示引用数据类型传参对对象属性的影响。
3. 编写一个方法 `modifyList`，接收一个 `ArrayList<Integer>` 作为参数，在方法内添加一个新元素到列表中。在 `main` 方法中创建 `ArrayList<Integer>` 对象，调用 `modifyList` 方法，并输出修改后的列表，验证引用数据类型传参对集合的影响。
4. 定义一个方法 `createCopy`，接收一个 `String` 数组作为参数，在方法内创建一个新的数组，将原数组的元素复制到新数组中，并返回新数组。在 `main` 方法中调用该方法，比较返回的新数组和原数组是否为同一个对象，体会方法返回引用类型对象的应用。
5. 编写一个方法 `processData`，接收一个整数和一个包含整数的 `HashSet` 作为参数。在方法内，将整数加 1，并向 `HashSet` 中添加一个新元素。在 `main` 方法中调用该方法，输出操作前后整数和 `HashSet` 的状态，分析基本数据类型和引用数据类型传参在该场景下的表现。
6. 定义一个类 `Book`，包含 `title` 和 `author` 属性。编写一个方法 `cloneBook`，接收一个 `Book` 对象作为参数，在方法内创建一个新的 `Book` 对象，复制原对象的属性，并返回新对象。在 `main` 方法中调用 `cloneBook` 方法，比较原对象和新对象的属性以及内存地址，理解方法返回引用类型对象实现对象复制的过程。
7. 编写一个方法 `changeArray`，接收一个整数数组作为参数，在方法内将数组的所有元素乘以 2。在 `main` 方法中创建一个整数数组，调用 `changeArray` 方法，并输出修改后的数组，展示引用数据类型传参时对数组元素的修改效果。
8. 定义一个方法 `updateObject`，接收一个自定义对象 `Person`（包含 `name` 和 `age` 属性）作为参数，在方法内创建一个新的 `Person` 对象，修改其属性后赋值给形参。在 `main` 方法中创建 `Person` 对象，调用 `updateObject` 方法，并输出原 `Person` 对象的属性，观察实参对象是否受到影响，理解引用数据类型传参时形参重新赋值的情况。
9. 编写一个方法 `copyList`，接收一个 `LinkedList<String>` 作为参数，在方法内创建一个新的 `LinkedList`，将原列表的元素复制到新列表中，并返回新列表。在 `main` 方法中调用该方法，验证新列表和原列表是否相互独立，体现方法返回引用类型对象在集合操作中的应用。
10. 定义一个方法 `operateData`，接收一个布尔值和一个包含布尔值的 `Vector` 作为参数。在方法内，将布尔值取反，并向 `Vector` 中添加一个新的布尔值。在 `main` 方法中调用该方法，输出操作前后布尔值和 `Vector` 的状态，分析基本数据类型和引用数据类型传参在不同数据结构下的特点。