Common Error 7.2

Uninitialized Arrays

A common error is to allocate an array reference, but not an actual array.

```java
double[] data;
data[0] = 29.95;  // Error—data not initialized
```

Array variables work exactly like object variables—they are only references to the actual array. To construct the actual array, you must use the `new` operator:

```java
double[] data = new double[10];
```

Advanced Topic 7.1

Array Initialization

You can initialize an array by allocating it and then filling each entry:

```java
int[] primes = new int[5];
primes[0] = 2;
primes[1] = 3;
primes[2] = 5;
primes[3] = 7;
primes[4] = 11;
```

However, if you already know all the elements that you want to place in the array, there is an easier way. List all elements that you want to include in the array, enclosed in braces and separated by commas:

```java
int[] primes = { 2, 3, 5, 7, 11 };
```

The Java compiler counts how many elements you want to place in the array, allocates an array of the correct size, and fills it with the elements that you specify.

If you want to construct an array and pass it on to a method that expects an array parameter, you can initialize an anonymous array as follows:

```java
new int[] { 2, 3, 5, 7, 11 }
```

7.2 Array Lists

The ArrayList class manages a sequence of objects.

Arrays are a rather primitive construct. In this section, we introduce the ArrayList class that lets you collect objects, just like an array does. Array lists offer two significant conveniences:
Array lists can grow and shrink as needed.

The `ArrayList` class supplies methods for many common tasks, such as inserting and removing elements.

Let us define an array list of bank accounts and fill it with objects. (The `BankAccount` class has been enhanced from the version in Chapter 3. Each bank account has an account number.)

```java
ArrayList<BankAccount> accounts = new ArrayList<BankAccount>();
accounts.add(new BankAccount(1001));
accounts.add(new BankAccount(1015));
accounts.add(new BankAccount(1022));
```

The type `ArrayList<BankAccount>` denotes an array list of bank accounts. The angle brackets around the `BankAccount` type tell you that `BankAccount` is a type parameter. You can replace `BankAccount` with any other class and get a different array list type. For that reason, `ArrayList` is called a generic class. You will learn more about generic classes in Chapter 17. For now, simply use an `ArrayList<T>` whenever you want to collect objects of type `T`. However, keep in mind that you cannot use primitive types as type parameters — there is no `ArrayList<int>` or `ArrayList<double>`.

When you construct an `ArrayList` object, it has size 0. You use the `add` method to add an object to the end of the array list. The size increases after each call to `add`. The `size` method yields the current size of the array list.

To get objects out of the array list, use the `get` method, not the `[ ]` operator. As with arrays, index values start at 0. For example, `accounts.get(2)` retrieves the account with index 2, the third element in the array list:

```java
BankAccount anAccount = accounts.get(2);
```

As with arrays, it is an error to access a nonexistent element. The most common bounds error is to use the following:

```java
int i = accounts.size();
anAccount = accounts.get(i); // Error
```

The last valid index is `accounts.size() - 1`.

To set an array list element to a new value, use the `set` method.

```java
BankAccount anAccount = new BankAccount(1729);
accounts.set(2, anAccount);
```

This call sets position 2 of the `accounts` array list to `anAccount`, overwriting whatever value was there before.

The `set` method can only overwrite existing values. It is different from the `add` method, which adds a new object to the end of the array list.

You can also insert an object in the middle of an array list. The call `accounts.add(i, a)` adds the object `a` at position `i` and moves all elements by one position, from the current element at position `i` to the last element in the array list.
After each call to the `add` method, the size of the array list increases by 1 (see Figure 3).

Conversely, the call `accounts.remove(i)` removes the element at position `i`, moves all elements after the removed element down by one position, and reduces the size of the array list by 1 (see Figure 4).

The following program demonstrates the methods of the `ArrayList` class. Note that you import the generic class `java.util.ArrayList`, without the type parameter.

```java
import java.util.ArrayList;

/**
 * This program tests the `ArrayList` class.
 */
public class ArrayListTester {
```
```java
public static void main(String[] args)
{
    ArrayList<BankAccount> accounts
        = new ArrayList<BankAccount>();
    accounts.add(new BankAccount(1001));
    accounts.add(new BankAccount(1015));
    accounts.add(new BankAccount(1729));
    accounts.add(1, new BankAccount(1008));
    accounts.remove(0);
    System.out.println("Size: " + accounts.size());
    System.out.println("Expected: 3");
    BankAccount first = accounts.get(0);
    System.out.println("First account number: "+ first.getAccountNumber());
    System.out.println("Expected: 1008");
    BankAccount last = accounts.get(accounts.size() - 1);
    System.out.println("Last account number: "
        + last.getAccountNumber());
    System.out.println("Expected: 1729");
}
```

**ch07/arraylist/BankAccount.java**

```java
/**
 * A bank account has a balance that can be changed by
 * deposits and withdrawals.
 */

public class BankAccount
{

    /**
     * Constructs a bank account with a zero balance.
     * @param anAccountNumber the account number for this account
     */
    public BankAccount(int anAccountNumber)
    {
        accountNumber = anAccountNumber;
        balance = 0;
    }

    /**
     * Constructs a bank account with a given balance.
     * @param anAccountNumber the account number for this account
     * @param initialBalance the initial balance
     */
    public BankAccount(int anAccountNumber, double initialBalance)
    {
        accountNumber = anAccountNumber;
        balance = initialBalance;
    }
```
/**
 * Gets the account number of this bank account.
 * @return the account number
 */
public int getAccountNumber()
{
    return accountNumber;
}

/**
 * Deposits money into the bank account.
 * @param amount the amount to deposit
 */
public void deposit(double amount)
{
    double newBalance = balance + amount;
    balance = newBalance;
}

/**
 * Withdraws money from the bank account.
 * @param amount the amount to withdraw
 */
public void withdraw(double amount)
{
    double newBalance = balance - amount;
    balance = newBalance;
}

/**
 * Gets the current balance of the bank account.
 * @return the current balance
 */
public double getBalance()
{
    return balance;
}

private int accountNumber;
private double balance;

Output
Size: 3
Expected: 3
First account number: 1008
Expected: 1008
Last account number: 1729
Expected: 1729
Self Check

3. How do you construct an array of 10 strings? An array list of strings?

4. What is the content of names after the following statements?

```java
ArrayList<String> names = new ArrayList<String>();
names.add("A");
names.add(0, "B");
names.add("C");
names.remove(1);
```

Common Error 7.3

Length and Size

Unfortunately, the Java syntax for determining the number of elements in an array, an array list, and a string is not at all consistent. It is a common error to confuse these. You just have to remember the correct syntax for every data type.

<table>
<thead>
<tr>
<th>Data Type</th>
<th>Number of Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Array</td>
<td>a.length</td>
</tr>
<tr>
<td>Array list</td>
<td>a.size()</td>
</tr>
<tr>
<td>String</td>
<td>a.length()</td>
</tr>
</tbody>
</table>

Quality Tip 7.1

Prefer Parameterized Array Lists

Parameterized array lists, such as `ArrayList<BankAccount>`, were introduced to the Java language in 2004. Versions of Java prior to version 5.0 had only an untyped class `ArrayList`. The untyped array list can hold elements of any class. (Technically, it holds elements of type `Object`, the "lowest common denominator" of all Java classes.) Whenever you retrieve an element from an untyped array list, the compiler requires you to use a cast:

```java
ArrayList accounts = new ArrayList();  // Untyped ArrayList
accounts.add(new BankAccount(1729));  // OK—can add any object
BankAccount a = (BankAccount) a.get(0);  // Need cast
```

The cast is needed because the compiler does not keep track of the objects that were inserted into the array list, and the array list `get` method has return type `Object`.

Untyped array lists are still a part of the Java language—after all, we want to continue to use programs that were written before 2004. But you should not use them for new code. The casts are tedious and also a bit error-prone. If you apply the wrong cast, the compiler cannot detect your mistake. Instead, your program will throw an exception.