

Review Exercises**R3.2**

- `dm = m * ((Math.sqrt(1 + b / c) / Math.sqrt(1 - v / c)) - 1);`

$$dm = m ((1 + b/c)^{1/2} / (1 - v/c)^{1/2}) - 1$$

- `volume = 4 * Math.PI * r * r * h;`

$$volume = \text{PI } r^2 h$$

- `volume = 4 * Math.PI + Math.pow(r,3) / 3;`

$$volume = 4 \text{ PI } r^3 / 3$$

- `p = Math.atan2(z, Math.sqrt(x * x + y * y));`

$$p = \tan^{-1} z (x^2 + y^2)$$

R3.3

The numerator is divided by 2, then multiplied by a, rather than divided by (2 * a).
 Parentheses in the denominator fix the problem:

```
x1 = (-b - Math.sqrt(b * b - 4 * a * c)) / (2 * a);
x2 = (-b + Math.sqrt(b * b - 4 * a * c)) / (2 * a);
```

R3.4

Give an example of integer overflow.

```
int i = 987654321 * 1000;
results in i being set to -18857080
```

Would the same example work correctly if you used floating-point ?

No. It becomes 9.87654321E11

Give an example of floating-point roundoff error.

4.35 * 100 becomes 434.99999999999994

Would the same example work correctly if you used integers ?

When using integers, you would of course need to switch to a smaller unit, such as cents instead of dollars or milliliters instead of liters.

The example would work correctly. The computation 435 * 100 yields 43500

R3.5

```
/**
 * This program tests the Purse class.
 */

import java.text.*;

public class PurseTest
{
    public static void main(String[] args)
    {
        Purse myPurse = new Purse();

        myPurse.addNickels(3);

        myPurse.addDimes(2);

        myPurse.addQuarters(1);

        NumberFormat formatter =
            NumberFormat.getCurrencyInstance();

        System.out.println(formatter.format(myPurse.getTotal()));
    }
}
```

The program prints the total as 0.6000000000000001 because it is of type double. The user may want to use the `NumberFormat` class in the `java.text` package, which can be used to format a number to any amount of decimal places. Specifically, the `getCurrencyInstance` method of the `NumberFormat` class can be used to generate a currency value. See Advance Topic 3.5.

R3.6

The statement

```
n = (int)x;
```

discards the fractional part of x . The statement

```
n = (int)Math.round(x);
```

rounds x to the nearest integer.

R3.7

In the first assignment,

```
n = (int)(x + 0.5)
```

0.5 is added to x before being converted to an integer.

In the second assignment,

```
n = (int)Math.round(x);
```

x will either be rounded up or down.

The difference between the two is that the first assignment will not work for $x < 0$.

For example, if $x = -0.99$, then $x + 0.5$ is -0.49 which gets rounded to 0 instead of -1.

R3.8

2 and 2.0 are numbers, where as "2" and "2.0" are strings containing the characters

that represent them. 2 is an integer and 2.0 is a floating-point value.

"2" is a string

of length 1, "2.0" is a string of length 3

R3.9

```
x = 2;  
y = x + x;
```

Computes $y = 2 + 2 = 4$

```
s = "2";  
t = s + s;
```

Computes $t = "2" + "2" = "22"$

R3.10

It is wasteful to initialize variables with 0 if the 0 is never used. It is better to move the variable definition to the point where you can fill the variable with its initial value.

R3.11

(a) `Integer.parseInt("" + x)` is the same as `x`.

True. `x` is converted to a string, then concatenated to the empty string. `parseInt` converts that string back to an integer, which is the same as the original value of `x`.

(b) `"" + Integer.parseInt(s)` is the same as `s`.

False. If `s` doesn't contain a number, then `parseInt` will fail.

Even if `s` contains a number, the resulting string can be different.

e.g. `s = "0.50"` yields the integer 0.5, which turns into the string "0.5",

which is a different string

(c) `s.substring(0, s.length())` is the same as `s`.

True. We are extracting all characters of `s`.

R3.12

How do you get the first character of a string?

```
String first_char = s.substring(0,1);
```

How do you get the last character of a string?

```
String last_char = s.substring(s.length() - 1, s.length());
```

How do you remove the first character of a string?

```
String no_first_char = s.substring(1,
s.length()); s = no_first_char;
```

How do you remove the last character of a string ?

```
String no_last_char = s.substring(0, s.length() - 1);
s = no_last_char;
```

R3.13

```
int last = n % 10;
```

The idea to get the first number is to compute 10 raised to the $\log(n)$ power, then divide. However, `Math.log` computes the natural log \ln , so we must get the decimal log as $\ln(x)/\ln(10)$.

```
double logn = Math.log(n) / Math.log(10);
int ndigits = (int)logn;
int pow10 = (int)Math.pow(10, ndigits);
int first = n / pow10;
```

R3.14

Give variables and constants meaningful names, something that describes their purpose or use. Initialize variables when defining them. Use constants instead of magic numbers.

R3.15

A `final` variable is a variable whose contents cannot be changed once it is initialized. That is, the value is constant.

You can actually define a `final` variable without supplying its value.

```
final int N;
// other statements . . .
N = 5; // after this initialization, N cannot change again
```

But this is not good practice. Just supply the initialization value when you define the `final` variable.

R3.16

(a) $x + n * y - (x + n) * y$
 $2.5 + (4 * -1.5) - (2.5 + 4) * -1.5 = 6.25$

(b) $m / n + m \% n = 6$

(c) $5 * x - n / 5 = 12.5$

(d) `Math.sqrt(Math.sqrt(n));`
1.414

```
(e) (int)Math.round(x) = 3

(f) (int)Math.round(x) + (int)Math.round(y) = 2;

(g) s + t;
"HelloWorld"

(h) s + n;
"Hello4"

(i) 1 - (1 - (1 - (1 - n))) = 4

(j) s.substring(1,3)
s is "ell"

(k) s.length() + t.length() = 10
```

R3.17

Copying numbers and copying object references uses the same type of commands. For example,

```
double balance1 = 1000;
double balance2 = balance1;
balance2 = balance2 + 500;
BankAccount account1 = new BankAccount(1000);
BankAccount account2 = account1;
account2.deposit(500);
```

The difference in this code is that the number variables hold values and the object variables hold references to another object. This means that the number variables are independent of each other once they are copied. The object variables, on the other hand, refer to each other, meaning that changing one variable results in changing the its reference also.

The result of copying the number variables above is:

```
balance1 = 1000;
balance2 = 1500;
```

The result of copying the object variables is:

```
account1 = 1500;
account2 = 1500;
```

R3.18

```
a = 2.0
```

```
a++ will add 1.0 + 1.0  
b = 1.0
```

same as `a` before the command `a++` because `a` and `b` are numeric variables and copying are independent

```
c = 0.25
```

by adding 5 nickels to the total

```
d = 0.25
```

since `q = p`, the call to `q.getTotal()` will yield the same results as `p.getTotal()`. `c` and `d` are numeric variables

that refer to object references. Therefore `c` and `d` get the same values

R3.19

It is not a problem in sharing string references because strings are immutable. None of the methods of the `String` class change the state of the the `String` object.