Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

SKU: 9780134477367-SOLUTIONS

Exercise 2.2

Zero.

Exercise 2.3

If too much money is inserted the machine takes it all - no refund.
If there isn't enough money inserted, it still prints out the ticket.

Exercise 2.5

It looks almost completely the same. Only the price on the ticket is different.

Exercise 2.6

The outer part of the student class:

```
public class Student
{
}
```

The outer part of the LabClass class:

```
public class LabClass
{
}
```

Exercise 2.7

Yes, the order of public and class matters.

Exercise 2.8

It is possible to leave out the word *public*.

Exercise 2.9

It is not possible to leave out the word *class*.

Exercise 2.10

Fields:
price
balance
total

Constructors:
TicketMachine

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

Methods:
getPrice
getBalance
insertMoney
printTicket

Exercise 2.11

It does not have any return type. The name of the constructor is the same as the name of the class.

Exercise 2.12

int
Student
Server

Exercise 2.13

alive
tutor
game

Exercise 2.14

`Student`, `Server`, `Person` and `Game`

Exercise 2.15

The exact order matters.

Exercise 2.16

Yes, it always necessary to have a semicolon after a field declaration.

Exercise 2.17

```
private int status;
```

Exercise 2.18

It belongs to the class Student.

Exercise 2.19

It has two parameters. One is of type String and the other of type double.

Exercise 2.20

It would be reasonable to expect the types to be the same as the two parameters (String and double).

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

Can't really assume anything about the names, but probably something like title and price.

Exercise 2.21

```
name = petsName;
```

Exercise 2.22

```
public Date(String month, int day, int year)
```

Exercise 2.23

Aside from their names, the only difference is that getPrice() returns the value of the price field whereas getBalance() returns the value of the balance field.

Exercise 2.24

How much money have I inserted into the machine?

Exercise 2.25

No. There is no direct link between the name of a method and the name of the field. However, it is a convention to use a name for the method that clearly links it to the field.

Exercise 2.26

```
public int getTotal()
{
    return total;
}
```

Exercise 2.27

Missing return statement.

Exercise 2.28

The header for getPrice() has an int as return type.
The header for printTicket() has void as return type.

Exercise 2.29

No.
Because they don't need to return anything.
Yes. Both headers have void as return types.

Exercise 2.31

It has a return type (void) and constructors do not have return types. It also does not have the same name as the class it is in.

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

Exercise 2.32

```
price = cost;
```

Exercise 2.33

```
score = score + points;
```

Exercise 2.34

It is a mutator.

Use an inspector to view the current score, then call the `increase` method with a positive parameter value and observe that `score` increases by that value.

Alternatively, if `score` has an accessor method; call the accessor, then call `increase`, and then call the accessor once again to verify that it returns the updated value, indicating that the field has been modified.

Exercise 2.35

```
price = price - amount;
```

Exercise 2.36
Note that no quote marks are printed, just the following:

```
My cat has green eyes.
```

Exercise 2.37

```
public void prompt()
{
    System.out.println("Please insert the correct amount of
money.");
}
```

Exercise 2.38

Instead of printing out the actual price of the ticket, it displays the word "price":

```
# price cents.
```

Exercise 2.39

Prints out the exact same string as in exercise 2.38

Exercise 2.40

No, because neither uses the value of the price field in the println statement.

Exercise 2.41

```
public void showPrice()
{
    System.out.println("The price of a ticket is " + price + "
cents.");
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
}
```

Exercise 2.42

They display different prices. This is because each ticket machine object has its own price field.
The price that was set in one ticket machine does not affect the other ticket machine's price.
The unique value of each ticket machine's price field is substituted into the println statement when
the method is called.

Exercise 2.43

```
public TicketMachine()
{
    price = 1000;
    balance = 0;
    total = 0;
}
```

When constructing a TicketMaching object you will not be prompted for a parameter value.
The tickets always have a price of 1000 cents.

Exercise 2.44

```
public TicketMachine()
{
    price = 1000;
    balance = 0;
    total = 0;
}

public TicketMachine(int ticketCost)
{
    price = ticketCost;
    balance = 0;
    total = 0;
}
```

Exercise 2.45

```
public void empty()
{
    total = 0;
}
```

It needs no parameters.
It is a mutator.

Exercise 2.46

The balance does not change when an error message is printed.
Inserting zero results in an error message.

Exercise 2.47

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

This version does not print an error message when zero is inserted.
Other than that, it does not change the observable behavior of the method.

Exercise 2.48
Note the importance of using <= rather than < in the rewritten condition.

```
if(amount <= 0) {
    System.out.println("Use a positive amount rather than: " +
                       amount);
}
else {
    balance = balance + amount;
}
```

Exercise 2.49

The field is: isVisible.
It determines whether the circle was visible or not.
Yes. As a circle is either visible or not, only two states (values) are needed.

Exercise 2.50

In the printTicket method of Code 2.8 the total is increased only by the price of the ticket, and not the full balance.
The balance is then decreased by the price.

Exercise 2.51
The else and associated block can be removed – if statements do not have to have an else part.

If an illegal amount is used, no error message is printed but the method otherwise works correctly.

Exercise 2.52

It could never become negative. The value in balance is checked in printTicket to ensure that it is always at least as large as price, so when price is subtracted balance cannot become negative.

Exercise 2.54

```
saving = price * discount;
```

Exercise 2.55

```
mean = total / count;
```

Exercise 2.56

```
if(price > budget) {
    System.out.println("Too expensive.");
}
else {
    System.out.println("Just right.");
}
```

Exercise 2.57

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
if(price > budget) {
    System.out.println("Too expensive. Your budget is only: " +
budget);
}
else {
    System.out.println("Just right.");
}
```

Exercise 2.58

Because balance is set to zero and then this new value is returned rather than the old one. The method will always return zero. It can be tested by inserting an amount, and then calling refundBalance(). The original would then return the amount inserted, but the new method returns 0.

Exercise 2.59

An error is printed: unreachable statement.

A return statement ends (exits) the method. Code after a return statement can therefore never be executed.

Exercise 2.60

The variable price is being re-declared as a local variable in the constructor – this is the effect of the word int in front of it. This local variable 'hides' the field of the same name. So the ticket price is never assigned to the price field.

Exercise 2.61

```
public int emptyMachine()
{
    int oldTotal = total;
    total = 0;
    return oldTotal;
}
```

Exercise 2.62

```
public void printTicket()
{
    int amountLeftToPay = price - balance;

    if(amountLeftToPay <= 0) {
        // Simulate the printing of a ticket.
        System.out.println("##################");
        System.out.println("# The BlueJ Line");
        System.out.println("# Ticket");
        System.out.println("# " + price + " cents.");
        System.out.println("##################");
        System.out.println();
        // Update the total collected with the price.
        total += price;
       // Reduce the balance by the prince.
        balance -= price;
    }
    else {
        System.out.println("You must insert at least: " +
                           amountLeftToPay + " more cents.");
    }
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
}
```

Exercise 2.63

You would need fields to store the prices of each of the tickets that the machine can issue.

You would need a method to select which type of ticket you would want.

It will not be necessary to modify many of the existing methods, if the price field is updated each time you select a new ticket type. You would probably need to modify the constructor, to allow several ticket prices.

Exercise 2.64

Name: getCode
Return type: String

Exercise 2.65

Name: setCredits
Parameter name: creditValue
Parameter type: int

Exercise 2.66

```
public class Person
{
}
```

Exercise 2.67

```
private String name;
private int age;
private String code;
private int credits;
```

Exercise 2.68

```
public Module(String moduleCode)
{
    code = moduleCode;
}
```

Exercise 2.69

```
public Person(String myName, int myAge)
{
    name = myName;
    age = myAge;
}
```

Exercise 2.70

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

The return type should not be void.

```
public int getAge()
{
    return age;
}
```

Exercise 2.71

```
public String getName()
{
    return name;
}
```

Exercise 2.72

```
public void setAge(int newAge)
{
    age = newAge;
}
```

Exercise 2.73

```
public void printDetails()
{
    System.out.println("The name of this person is " + name);
}
```

Exercise 2.74

student1 : Student
name: Benjamin Jonson
id: 738321
credits: 0

Exercise 2.75

"Henr557"

Exercise 2.76

It opens the editor with the source code for the Student class. It displays a message: StringIndexOutOfBoundsException
This happens because the method getLoginName expects the name to be at least 4 characters long and "djb" is only 3 characters.

Exercise 2.77

```
public Student(String fullName, String studentID)
{
    if(fullName.length() < 4) {
        System.out.println("Error! The name should be at least 4
characters long");
    }
    if(studentID.length() < 3) {
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
        System.out.println("Error! The studentID should be at
least 3 characters long");
    }
    name = fullName;
    id = studentID;
    credits = 0;
}
```

Exercise 2.78

```
public String getLoginName()
{
    String loginNamePart;
    if(name.length() < 4) {
        loginNamePart = name;
    }
    else {
        loginNamePart = name.substring(0,4);
    }

    String loginIdPart;
    if(id.length() < 3) {
        loginIdPart = id;
    }
    else {
        loginIdPart = id.substring(0,3);
    }

    return  loginNamePart + loginIdPart ;
}
```

Exercise 2.79

102
"catfish"
"cat9"
"12cat"
"cat39"
"f"
StringIndexOutOfBoundsException

Exercise 2.80

The first call returns 0 and the second returns 500.

Exercise 2.81

Because t2 refers to the same object as t1, the call will print 500.

This example of *aliasing* is an important one and students should try to ensure that they understand what is going on here.

Exercise 2.82

The call returns 1000. Even though the change was made via t1, because t2 is referring to the same object, it sees the new value. Note that we have only created a single TicketMachine object in these two exercises, but two variables refer to that one object.

Exercise 2.83

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
/**
 * Returns the author of this book.
 */
public String getAuthor()
{
    return author;
}

/**
 * Returns the title of this book.
 */
public String getTitle()
{
    return title;
}
```

Exercise 2.84

```
/**
 * Prints the name of the author in the terminal window.
 */
public void printAuthor()
{
    System.out.println("Author: " + author);
}

/**
 * Prints the title of the book in the terminal window.
 */
public void printTitle()
{
    System.out.println("Title: " + title);
}
```

Exercise 2.85

Delete the constructor and insert this:

```
private int pages;

/**
 * Set the author and title fields when this object
 * is constructed.
 */
public Book(String bookAuthor, String bookTitle, int bookPages)
{
    author = bookAuthor;
    title = bookTitle;
    pages = bookPages;
}

/**
 * Returns the number of pages in this book.
 */
public int getPages()
{
    return pages;
}
```

Exercise 2.86

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

The objects are immutable because the class contains no methods to change the values of any of the fields once an instance has been created.

Exercise 2.87

```
public void printDetails()
{
    System.out.print ("Title:  " + title + ", ");
    System.out.print("Author: " + author + ", ");
    System.out.println("Pages:  " + pages);
}
```

Exercise 2.88

Delete the constructor and insert:

```
private String refNumber;

/**
 * Set the author and title fields when this object
 * is constructed.
 */
public Book(String bookAuthor, String bookTitle, int bookPages)
{
    author = bookAuthor;
    title = bookTitle;
    pages = bookPages;
    refNumber = "";
}

/**
 * Sets the reference number for this book
 */
public void setRefNumber(String ref)
{
    refNumber = ref;
}

/**
 * Gets the reference number for this book
 */
public String getRefNumber()
{
    return refNumber;
}
```

Exercise 2.89

```
public void printDetails()
{
    System.out.println("Title:  " + title);
    System.out.println("Author: " + author);
    System.out.println("Pages:  " + pages);

    String refNumberString;
    if(refNumber.length() > 0 ) {
        refNumberString = refNumber;
    }
    else {
        refNumberString = "ZZZ";
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
    }
    System.out.println("Reference number:  " + refNumberString);
}
```

Exercise 2.90

```
public void setRefNumber(String ref)
{
    if(ref.length() >= 3) {
        refNumber = ref;
    }
    else {
        System.out.println("Error! The reference number must be at
least 3 characters long.");
    }
}
```

Exercise 2.91

Add the field:

```
private int borrowed;
```

Add the methods:

```
/**
 * Borrows the book. Increments the number of times the book has
been borrowed.
 */
public void borrow()
{
    borrowed++;
}

/**
 * Gets the number of times the book has been borrowed.
 */
public int getBorrowed()
{
    return borrowed;
}
```

Add this line to printDetails method:

```
System.out.println("Borrowed: " + borrowed);
```

Exercise 2.92

```
private boolean courseText;

public boolean isCourseText()
{
    return courseText;
}
```

Exercise 2.93

```
public class Heater
{
    private double temperature;
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
    /**
     *  Creates a new Heater with an initial temperature of 15.
     */
    public Heater()
    {
        temperature = 15.0;
    }

    /**
     * Increases the temperature by 5 degrees
     */
    public void warmer()
    {
        temperature += 5.0;
    }

    /**
     * Decreases the temperature by 5 degrees
     */
    public void cooler()
    {
        temperature -= 5.0;
    }

    /**
     * Gets the current temperature of the heater
     */
    public double getTemperature()
    {
        return temperature;
    }
}
```

Exercise 2.94

```
public class Heater
{
    private double temperature;
    private double min;
    private double max;
    private double increment;

    /**
     *  Creates a new Heater with an initial temperature of 15.
     */
    public Heater(double minimum, double maximum)
    {
        min = minimum;
        max = maximum;
        temperature = 15.0;
        increment = 5.0;
    }

    /**
     * Increases the temperature
     */
    public void warmer()
    {
        double newTemperature = temperature + increment;
```

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual

```
        if(newTemperature <= max) {
           temperature = newTemperature;
        }
    }

    /**
     * Decreases the temperature
     */
    public void cooler()
    {
        int newTemperature = temperature - increment;
        if(newTemperature >= min) {
            temperature = newTemperature;
        }
    }

    /**
     * Sets the increment, which determines how much the two
methods
     * warmer() and cooler() changes the temperature.
     */
    public void setIncrement(double inc)
    {
        if(inc >= 0) {
            increment = inc;
        }
    }

    /**
     * Gets the current temperature of the heater
     */
    public double getTemperature()
    {
        return temperature;
    }
}
```

If the setIncrement method does not check for negative values, and a negative value is passed, then the program will not work as expected. The minimum and maximum values can be exceeded.

Objects First with Java A Practical Introduction Using BlueJ 6th edition Barnes Solutions Manual