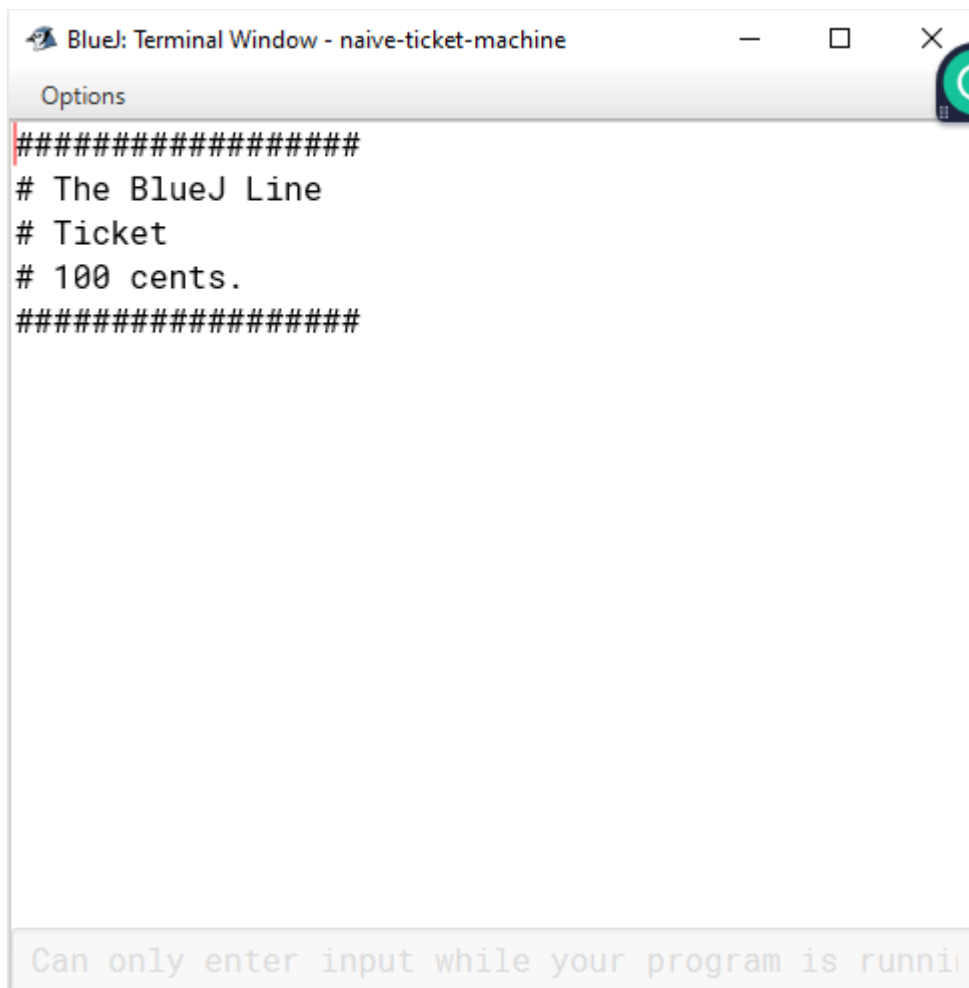


Assignment 1 Book Exercises

Exercise 2.1:

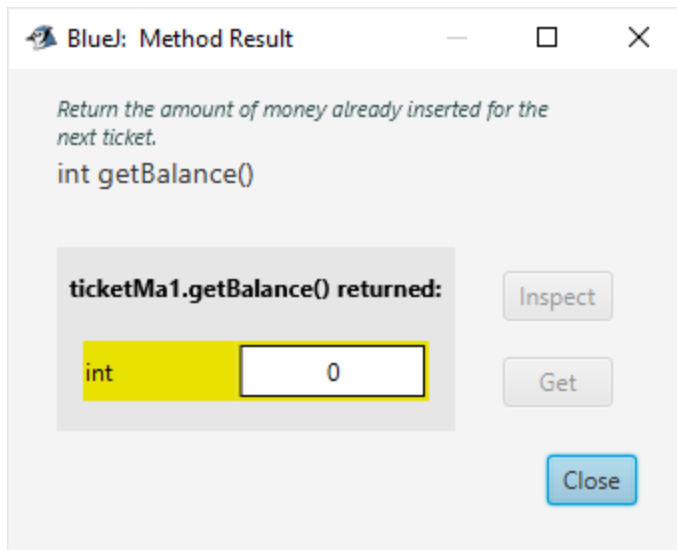


The screenshot shows a BlueJ Terminal Window titled "BlueJ: Terminal Window - naive-ticket-machine". The window has a standard macOS-style title bar with minimize, maximize, and close buttons. Below the title bar is a tab labeled "Options". The main area of the terminal displays the following text:

```
#####  
# The BlueJ Line  
# Ticket  
# 100 cents.  
#####
```

At the bottom of the terminal, there is a light gray message box that reads: "Can only enter input while your program is running".

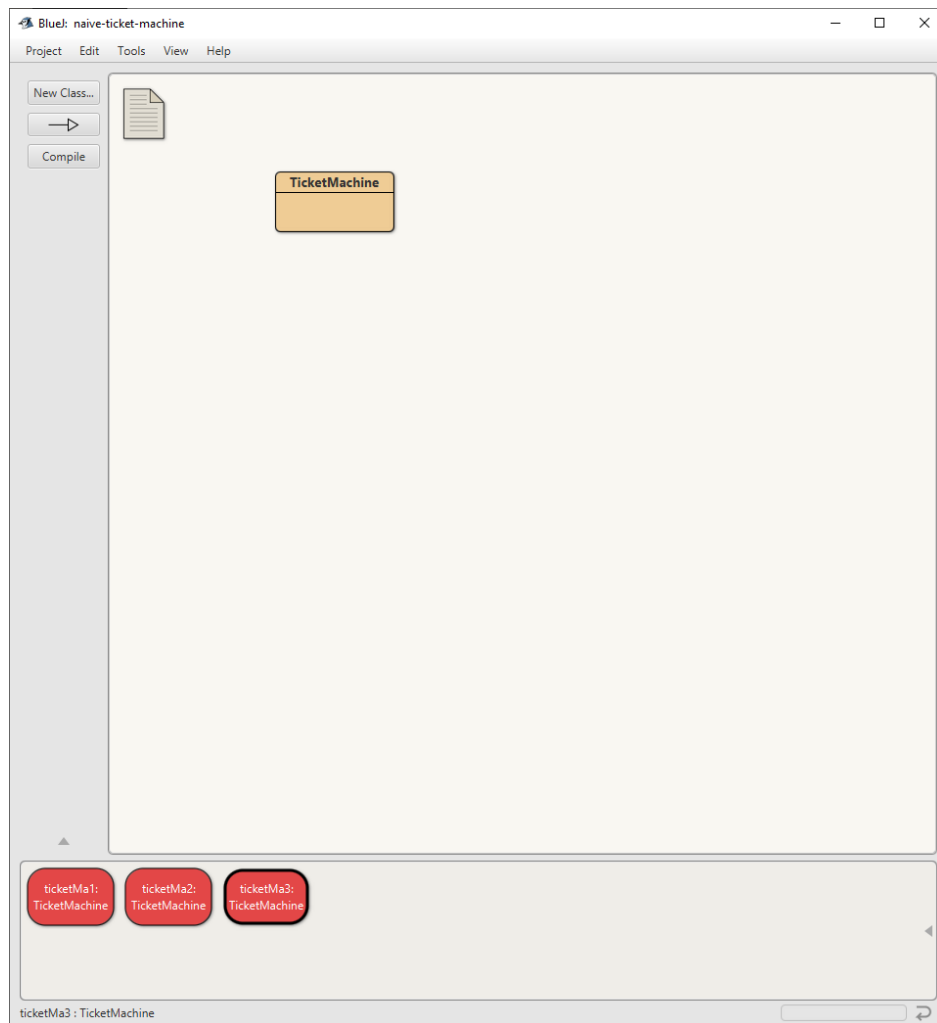
Exercise 2.2:



Exercise 2.3:

The machine prints the ticket with the assumption that it was paid for -- hence the name 'naïve ticket machine'.

Exercise 2.4-5:



Exercise 2.6:

```
public class Student
{
}

public class LabClass
{
}
```

Exercise 2.7:

Yes, it matters.

```

public class TicketMachine
{
    // The price of a ticket from this
    private int price;
    // The amount of money entered by

```

Exercise 2.8:

It is possible to declare a class without specifying an access modifier, in which case it is implicitly public. This is not the best practice.

Exercise 2.9:

It will not be compiled without specifying the data type, as the 'Ticket Machine' is not a declared type yet.

Exercise 2.10:

Constructors:

```

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

```

Fields:

```

// The price of a ticket from this machine.
private int price;
// The amount of money entered by a customer so far.
private int balance;
// The total amount of money collected by this machine.
private int total;

```

Methods:

```

public int getPrice()
public int getBalance()
public void insertMoney(int amount)
public void printTicket()

```

Exercise 2.11:

It is named the same as the class and does not return a value.

Exercise 2.12:

Declaration:

Type:

private int count;	Integer
private Student representative;	Student
private Server host;	Server

Exercise 2.13:

Field:

Name:

private boolean alive;	alive
private Person tutor;	Tutor
private Game game;	game

Exercise 2.14:

Student, Server, Person, and Game are all class names.

Exercise 2.15:

Yes, it matters.



Exercise 2.16:

Yes, semicolons are required.

Exercise 2.17:

public int status;

Exercise 2.18:

The Student class.

Exercise 2.19:

2 parameters. 1 String and 1 double.

Exercise 2.20:

I would guess the Book class would have the following (and more) fields:

Pages, Author, Publisher, ISBN, etc.

Exercise 2.21:

```
public Pet(String petsName)
{
    this.name = petsName;
}
```

Exercise 2.22:

```
public class Date()
{
    public Date(String month, int day, int year)
    {
    }
}
```

Exercise 2.23:

The getPrice() method returns the value of the price field, whereas getBalance() returns the value of the balance field.

Exercise 2.24:

“How much money has been inserted into the machine?”

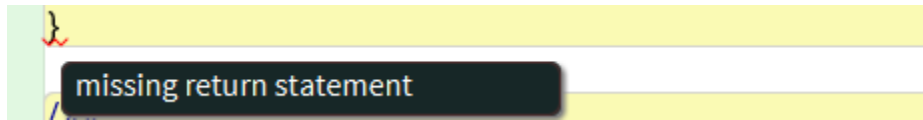
Exercise 2.25:

The name does not effect its function.

Exercise 2.26:

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

Exercise 2.27:



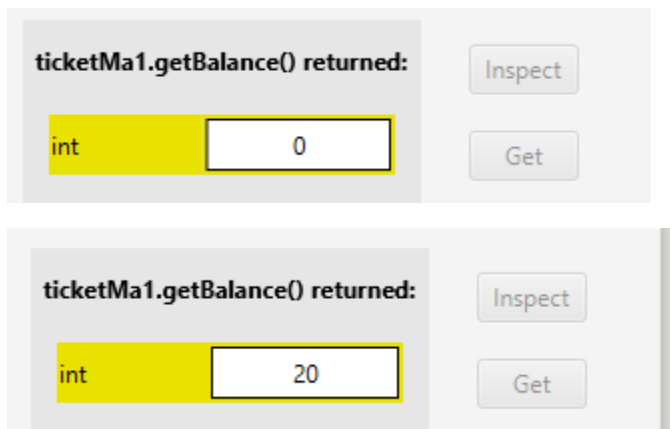
Exercise 2.28:

printTicket is a void, getPrice() returns an int.

Exercise 2.29:

They are voids because they are not used to access a specific value. They are mutator methods that alter the value of a private field.

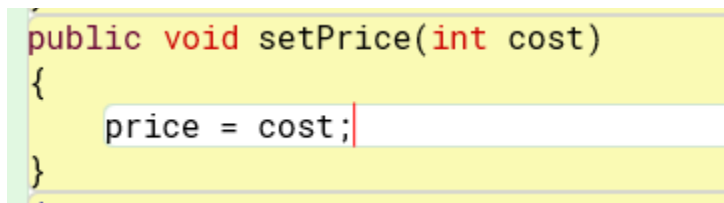
Exercise 2.30:



Exercise 2.31:

Because the return type is void.

Exercise 2.32:



Exercise 2.33:

```
public void increase(int points)
{
    score += points;
}
```

Exercise 2.34:

Yes, it is a mutator method of the score field.

Exercise 2.35:

```
public void discount(int amount)
{
    Price -= amount;
}
```

Exercise 2.36:

My cat has green eyes.

Exercise 2.37:

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

Exercise 2.38-39:

The output would be: '# price cents.'

Exercise 2.40:

Neither of them show the price of tickets, so no. It could be modified to output prices of different ticket machines with the appropriate parameters, however.

Exercise 2.41:

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

Exercise 2.42:


```
getPrice(int cost)

Blue: Terminal Window - naive-ticket-machine

Options

The price of a ticket is 100 cents.
The price of a ticket is 200 cents.
```

The outputs are different because the method is run on different objects with different values.

Exercise 2.43:

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

It no longer requires input to instantiate.

Exercise 2.44:

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

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

Exercise 2.45:

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

Exercise 2.46:

No, it does not.

Exercise 2.47:

The only time this will make a difference is when the input value results in the test failing. In such a case the method would do nothing.

Exercise 2.48:

```
public void insertMoney(int amount)
{
    if(amount <= 0)
    {
        System.out.println("Please insert a positive value");
    }
}
```

Exercise 2.49:

The fact that it can only be one of two values is fine if there is more than one test.

Exercise 2.50:

The better version only prints the ticket if enough money has been inserted.

Exercise 2.51:

No, the else statement is necessary here because there is no return nested in the if block, which would cause the error message to output to the console every time the method is executed.

Exercise 2.52:

Yes, if too much money was inserted

```
total = total + price;
// Reduce the balance by the price.
balance = balance - price;
}
```

Exercise 2.53:

Appendix C

Operators

Exercise 2.54:

```
public void mult()  
{  
    saving = price * discount;  
}
```

Exercise 2.55:

```
public void div()  
{  
    mean = total / count;  
}
```

Exercise 2.56:

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

Exercise 2.57:

```
if(price > budget)
{
    System.out.println("Too expensive, your budget is: " + budget);
}
else
```

Exercise 2.58:

Because it sets the value of balance to 0 before returning it.

Exercise 2.59:

It will not compile because there is code after the return statement within the same scope.

Exercise 2.60:

It creates a local variable named price, instead of modifying the global variable.

Exercise 2.61:

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

Exercise 2.62:

```

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 = total + price;
    // Reduce the balance by the price.
    balance = balance - price;
}
else {
    System.out.println("You must insert at least: " +
        amountLeftToPay + " more cents.");
}
}

```

Exercise 2.63:

A discount field that is used to calculate a total price. This would minimize the alterations required.

Exercise 2.64:

String.

Exercise 2.65:

The method name is setCredits, the parameter type is int.

Exercise 2.66:

```

public class Person()
{
    public Person()
    {
        |
    }
}

```

Exercise 2.67:

```
public class Person()
{
    private String name;
    private int age;
    private String code;
    private int credits;

    public Person()
    {
        this.name = 'name';
        this.age = 0;
        this.code = 'code';
        this.credits = 100;
    }
}
```

Exercise 2.68:

```
public class Module()
{
    private String code;

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

Exercise 2.69:

```

public class Person()
{
    private String name;
    private int age;
    private String code;
    private int credits;

    public Person(String myName, int myAge)
    {
        this.name = myName;
        this.age = myAge;
        this.code = 'code';
        this.credits = 100;
    }
}

```

Exercise 2.70:

```

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 the person is " + name);
}
```

Exercise 2.74:

Student1:

Student

Name:

ID:

Exercise 2.75:

Henry Moore

Exercise 2.76:

The output is 'djb'.

Exercise 2.77:

Done.

Exercise 2.78:

Set a default value so that there is always something returned.

Exercise 2.79:

Done.

Exercise 2.80:

Missing semicolons prevent the code from compiling.

Exercise 2.81:

The balance of the t2 object is returned.

Exercise 2.82:

The balance of t2 is not affected by the insertMoney method that was run on t1.

Exercise 2.83:


```

public String getAuthor()
{
    return author;
}
public String getTitle()
{
    return title;
}

```

Exercise 2.84:

```

public void printAuthor()
{
    System.out.println(author);
}
public void printTitle()
{
    System.out.println(title);
}

```

Exercise 2.85:

```

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

// Add the methods here ...
public int getPages()
{
    return pages;
}

```

Exercise 2.86:

Yes, because there are no mutator methods.

Exercise 2.87:

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

Exercise 2.88:

```
private int pages;
private String refNumber;
```

```
public void setRefNumber(String ref)
{
    refNumber = ref;
}
```

Exercise 2.89:

```
*/
public Book(String bookAuthor, String bookTitle, int pageCount, String ref)
{
    author = bookAuthor;
    title = bookTitle;
    pages = pageCount;
    refNumber = ref;
}
```

```
System.out.println("Title: " + title + " Author: " + author + " Pages: "
if (refNumber.length() >= 3)
{
    System.out.println("Ref. Number: " + refNumber);
}
```

Exercise 2.90:

```

public void setRefNumber(String ref)
{
    if (ref.length() >= 3)
    {
        refNumber = ref;
    }
    else
    {
        System.out.println("Error");
    }
}

```

Exercise 2.91:

```

public void borrow()
{
    borrowed += 1;
}

```

Exercise 2.92:

```

public Book(String bookAuthor, String bookTitle, int pageCount, boolean forCourse)
{
    author = bookAuthor;
    title = bookTitle;
    pages = pageCount;
    refNumber = reference;
    borrowed = 0;
    courseText = forCourse;
}

```

Exercise 2.93:

```
public class Heater()
{
    private double temperature;
    public Heater()
    {
        temperature = 15.0;
    }
    public void warmer()
    {
        temperature += 5.0;
    }
    public void cooler()
    {
        temperature -= 5.0;
    }
    public double getTemperature()
    {
        return temperature;
    }
}
```

Exercise 2.94:

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

    public Heater()
    {
        temperature = 15.0;
    }

    public void warmer()
    {
        if (temperature + increment >= max)
        {
            temperature = max;
            break;
        }

        temperature += increment;
    }

    public void cooler()
    {
        if (temperature - increment <= min)
        {
            temperature = min;
            break;
        }

        temperature -= increment;
    }

    public double getTemperature()
    {
        return temperature;
    }
}
```

