Chapter 2

2.1: Printed out a ticket after inserting the exact amount of money needed to buy a ticket

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

2.2: After printing the ticket and then calling getBalance the value 0 was returned

2.3: Inserting too much money and then printing ticket sets the balance to 0 ignoring the extra money. The ticket machine also still prints tickets when not enough money is put in and sets the balance to 0

2.4: *TicketMachine behavior understood*

2.5: TicketMachine the costs a different cost prints out a ticket with the different cost on it

```
 ##################
# The BlueJ Line
# Ticket
# 4000 cents.
##################
```

2.6: Outer wrappers for the student and Labclass classes might look like

```
public class student
{

}

public class LabClass
{

}
```

2.7: Yes, it matter we get error messages include

```
<identifier> expected
Invalid method declaration; return type required
```

The error messages are clear and show exactly where the errors are and what it expects

2.8: The code still compiles when public is left off

2.9: The code does not compile when class is left off and the error message is “class, interface or enum expected"

2.10: The fields for ticket machine include are price, balance, total. The only constructor is TicketMachine. And the methods are getPrice, getBalance, insertMoney, and printTicket

2.11: the constructor does not have the keyword class and it assigns all of the fields values

2.12: private int count; is of type int
Private Student representative; is of type Student
Private Server host; is of type Server
2.13: private boolean alive; alive is the name of the field
Private Person tutor; tutor is the name of the field
Private Game game; game is the name of the field
2.14: Knowing that class names usually start in uppercase letters so i would guess that Student, Server, Person, and Game are all classes
2.15: The order matters for the field declaration it won't work if it is not in order

2.16: There always needs to be a semicolon at end of field declaration
2.17: private int status

2.18: public Student(String name) the constructor belongs to the class Student
2.19: public Book(String title, double price) has 2 parameters one is a String and one is a double
2.20: The names of the fields will be different from the parameters so maybe it could the names could be name and cost

2.21: name = petsName
2.22: public Date(int month, int day, int year)

2.23: the only difference is from what field it the value from. getBalance return the value from the balance field while getPrice returns the value from the price field
2.24: “How much money is in the machine right now?”
2.25: The return statement does not need to be changed when the name of the method is changed
2.26:
```
public int getTotal()
        {
                Return total;
        }
```
2.27: Missing return statement
2.28: The getPrice method returns an int and the printTicket method returns nothing
2.29: The methods insertMoney and printTicket don't have return statements. We know this from the header and the word void. This is because they don't return information

2.30: They showed different outputs

2.31: A header for a constructor would never have the word void in it
2.32:
```
public void setPrice(int cost)
        {
                price = cost;
        }
```

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

2.34: The increase method is a mutator it alters its object

```
2.35: public void discount(int amount)
        {
                price = price - amount;
        }
```

2.36: My cat has green eyes

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

2.38: # price cents
2.39: # price cents
2.40: No, because there are no variables in either line

```
2.41: public void showPrice()
        {
                System.out.println(“The price of the ticket is ” + price + “ cents”);
        }
```

2.42: The showPrice method shows different prices for different machines because it returns the price variable which could be different depending on the ticket machine

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

Now the user is no longer prompted to enter a price when creating ticket machines
2.44: *Machines Tested*

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

This method doesn't need any parameters it is always changing the field to 0 and it is a mutator because it is changing the empty
2.46: The balance doesn't change if there is an error message printed. An error message is printed if zero is used as the parameter for the insertMoney method.
2.47: If we use the greater-than or equal-to operator then the insertMoney method will accept 0 as a value for its parameters

2.48: public void insertMoney(int amount)
```
	{
		 if (amount <= 0) {
		system.out.println("Use a positive amount");
		}
		else {
		balance = balance + amount;
		}
	}
```
2.49: the figures project uses boolean for the visibility of the object. This is apropriate because the object can be either visible or invisible.
2.50: The naive ticket machine increases the total with the balance and always sets the blance to zero. The better ticket machine checks the balance field to make sure its equal or greater than the price. Then the printTicket method reduces the balance by the price.
2.51: It is possible to compile without the else statement but you get no feedback if you try calling the printTicket method without enough in the balance to buy a ticket

2.52: No, the balance field cannot be negative because the printTicket method checks if there is enough in the balance is high enough before the value is subtracted
2.53: Other operators include
* multiplication
/ division
% modulus, or remainder after division
2.54: saving = price*discount;
2.55: mean = total/count;
2.56: if (price > budget){
```
	system.out.println("Too expensive");
	}
	Else {
	System.out.println("Just right");
	}
```
2.57: if (price > budget){
```
	system.out.println("Too expensive, current budget: " + budget);
	}
	Else {
	System.out.println("Just right");
	}
```

2.58: The code will always return the value of 0 because it sets it to 0 then returns it.
Testing include trying to get the method to produce a value other than 0
2.59:
2.60: The code compiles but doesn't run properly. The problem is the code stores the price in a local variable that gets deleted.

2.61:

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

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 balance.
    total = total + balance;
    //
    balance = balance - price;
  }
  else {
    System.out.println("Amount left to pay: " + amountLeftToPay + " cents.")
  }
  }
}
```

2.63: The constructor would need to be changed so the user isn't asked for a price

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

Then add a method that sets the price to whatever the discount should be

```
public void SetPrice(int cost)
  {
    price = cost;
  }
```

2.64: Name: getCode()
Return type: String

2.65: Method: setCredits
Parameter name: creditValue
Parameter type: integer

```
2.66: public class Person
        {

         }
```

2.67:

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

```
2.68: public Module(String moduleCode)
            {
            code = moduleCode;
            }
2.69: public Person(String myName, int myAge)
            {
            name = myName
            age =  myAge
            }
```

2.70: The method returns something but has a void return type the code is supposed to return an int to the void should be an int

```
public int getAge()
{
Return age;
}
2.71:public String getName()
            {
            Return name;
            }
2.72: public void setAge(int myAge)
            {
            age = myAge;
            }
2.73: public void printDetails()
            {
            System.out.println("The name of this person is " + name);
```

```
    }
```
2.74:

name	“Benjamin Jonson”
id	“738321”
credits	0

2.75: “henr557”
2.76: The user gets the following error message
java.lang.StringIndexOutOfBoundsException: String index out of range: 4
this is because the name is too short and the method needs a string with four characters
2.77:
```
public Student(String fullName, String studentID)
   {
      if(fullName.length() < 4){
         System.out.println("Name is too short");
      }

      name = fullName;
      id = studentID;
      credits = 0;
   }
```

2.78:
```
public String getLoginName()
      {
         if(name.length() > 4 && id.length() > 3) {
         return name.substring(0,4) + id.substring(0,3);
      }
         else{
           return name + id;
          }
      }
```

2.79: 102
“catfish”
“cat9”
“12cat”
“Cat39” i expected this one to be cat12
“F”
java.lang.StringIndexOutOfBoundsException: String index out of range: 8

2.80:t1.getBalance()   0 (int)

t1.insertMoney(500);
t1.getBalance()		500 (int)
2.81: TicketMachine t2 = t1 this copies our first ticket machine to our second ticket machine.
t2.getBalance() would return 500
2.82: I would not expect changing the balance t1 after the creation t2 to affect t2. However this does not work this way, when the insertMoney method is called for wither machine it changed the other.

2.83:
```
public String getAuthor()
{
        return author;
}
public String getTitle()
{
        return title;
}
```
2.84:
```
public void printAuthor()
    {
        System.out.println(author);
    }
    public void printTitle()
    {
        System.out.println(title);
    }
```
2.85:
```
//fields
   author = bookAuthor;
   title = bookTitle;
   pages = bookPages;
//constructor
    public Book(String bookAuthor, String bookTitle, int bookPages)
    {
        author = bookAuthor;
        title = bookTitle;
        pages = bookPages;
    }
//method
    public int getPages()
    {
        return pages;
    }
```

2.86: Yes, the book objects are immutable. The book objects have no methods that alter any of the fields so the book cannot be changed after its made

2.87:
```
   public void printDetails()
   {
      System.out.println("title: " + title + " Author: " + author + " pages: " + pages);
   }
```
2.88:
```
//fields
   private String author;
   private String title;
   private int pages;
   private String refNumber;
//constructor
   public Book(String bookAuthor, String bookTitle, int bookPages)
   {
      author = bookAuthor;
      title = bookTitle;
      pages = bookPages;
      refNumber = "";
   }
//methods
public void setRefNumber (String ref)
   {
      refNumber = ref;
   }

public String getRefNumber()
   {
      return refNumber;
   }
```
2.89:
```
public void printDetails()
   {
      System.out.println("title: " + title + " Author: " + author + " pages: " + pages);

      if(refNumber.length() < 1){
         System.out.println("ref number: " + "zzz");
      }
         else{
          System.out.println("ref number: " + refNumber);
         }
   }
```

2.90:
```
    public void setRefNumber (String ref)
    {
        if(ref.length() < 3){
            System.out.println("ref number needs to be at least 3 characters");
        }
        else{
        refNumber = ref;
        System.out.println("ref number set: " + ref);
    }
    }
```
2.91:
```
//fields
    private String author;
    private String title;
    private int pages;
    private String refNumber;
    private int borrowed;


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

public int getBorrowed()
    {
        return borrowed;
    }
public void printDetails()
    {
        System.out.println("title: " + title + " Author: " + author + " pages: " + pages);
        System.out.println("times borrowed: " + borrowed);
        if(refNumber.length() < 1){
            System.out.println("ref number: " + "zzz");
        }
            else{
             System.out.println("ref number: " + refNumber);
            }
    }
```
2.92:
```
//field
```

```
private boolean courseText;
//constructor
public Book(String bookAuthor, String bookTitle, int bookPages, boolean bookCourseText)
    {
        author = bookAuthor;
        title = bookTitle;
        pages = bookPages;
        refNumber = "";
        courseText = bookCourseText;
    }
//method
public boolean isCourseText()
    {
        return courseText;
    }
```

2.93:

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

    public Heater()
    {
        temperature = 15.0;
    }
    //mutators
    public void warmer()
    {
        temperature = temperature + 5.0;
    }
    public void cooler()
    {
        temperature = temperature - 5.0;
    }
    //accessor
    public double getTemp()
    {
        return temperature;
    }
}
```

2.94:

```
public class Heater
{
    //fields
    private double temperature;
    private double min;
    private double max;
    private double increment;
    //constructor
    public Heater(double minimum, double maximum)
    {
        temperature = 15.0;
        min = minimum;
        max = maximum;
        increment = 5.0;
    }
    //mutators
    public void warmer()
    {
        if(temperature < max){
        temperature = temperature + increment;
    }
        else{
            System.out.println("Temperature is already at max");
        }
    }
    public void cooler()
    {
        if(temperature > min){
        temperature = temperature - increment;
    }
        else{
            System.out.println("Temperature is already at min");
        }
    }
```

//rest on next page

```
  public void setIncrement(double Increment)
    {
       if(Increment > 0) {
       increment = Increment;
    }
       else {
          System.out.println("Please use a positive number");
       }
    }
    //accessor
    public double getTemp()
    {
       return temperature;
    }
}
```