

> Lesson Summary and Outcomes

In this unit, students will take a `VendingMachine` class and use if statements to add error checking. See sample `VendingMachine` class without error checking in the addendum. Students will add code to prevent certain actions from being taken, for example, blocking a user from entering a negative dollar amount or receiving a product before inserting enough money in the vending machine.

> Student Assessment

How well do students perform each of the following tasks:

- Enhance the `VendingMachine` class (see addendum) with conditionals to include error checking. Users should not be able to do the following:
 - Add a negative dollar amount
 - Add an amount exceeding some predetermined limit
 - Get a product until enough money has been inserted
 - Insert money if no products are available
- Comment everything
- Save, compile, and test often

> Prerequisite Knowledge and Skills

BlueJ:

- Creating projects
- Adding new classes
- Instantiating objects
- Testing methods from instantiated objects

Java:

- Class structure, including methods
- Conditionals

> This Lesson Targets the Following Subject Areas(s):

Pre-Java	Java Programming
<input type="checkbox"/> Hardware Basics	<input type="checkbox"/> Applet Programming
<input checked="" type="checkbox"/> Software Basics	<input type="checkbox"/> Subroutine Programming
<input type="checkbox"/> Networks and Servers	<input checked="" type="checkbox"/> Full Scale Programming
<input type="checkbox"/> HTML	
<input type="checkbox"/> Action Scripting	
<input type="checkbox"/> Java Scripting	

> Curriculum-Framing Questions

Essential Question

- In what situations are conditionals used?

Lesson Questions

- Where should conditionals be employed in the `VendingMachine` class?
- What are some of the error conditions that could arise?

Sample Content Questions

- Why should conditionals be used in this situation?
- What is essential to maintaining the integrity of the `VendingMachine` class?

> Targeted Content Standards, Benchmarks, or State Frameworks

Colorado State Standards:

Mathematics:

- **Standard 1** – Students develop number sense and use numbers and number relationships in problem-solving situations and communicate the reasoning used in solving these problems.
 - **Grades 9–12** – Use number sense to estimate and justify the reasonableness of

International Society of Technology Education (ISTE): Information Literacy Standards:

- **Standard 2** – The student who is information literate evaluates information critically and competently.

Technology Foundation Standards for Students:

- **Standard 6** – Technology

solutions to problems involving real numbers.

problem-solving and decision-making tools. Students use technology resources for solving problems and making informed decisions. Students employ technology in the development of strategies for solving problems in the real world.

> Materials and Resources Required for Lesson

Equipment	<ul style="list-style-type: none"> • Computer for each student with BlueJ and Java SDK installed
Consumable Supplies	<ul style="list-style-type: none"> • Diskettes • Printed output for prototype
Textbooks/Lesson Guides	<ul style="list-style-type: none"> • Objects First With Java: A Practical Introduction to BlueJ by David J. Barnes and Michael Kölling • Handouts of lessons
Technology	<ul style="list-style-type: none"> • Windows-based machines running Windows 98 SE or higher

> Lesson Plan Outline

Overview: Students will take a basic `VendingMachine` class and improve on it by using conditional (`if`) statements to prevent error conditions, such as attempts by users to enter a negative amount of money or to retrieve a product without depositing sufficient money. Students will use conditional statements as a means of determining when a method should proceed and when there is a problem. The basic format of a conditional statement is as follows:

```
if (some_expression_is_true)
{
    // execute the appropriate commands
}
```

The method will execute the commands that exist between the brackets `{ }` only when the expression on the `if` line is true. When the expression is false, those commands will be ignored. You can extend conditionals in several ways. You can also check more than one expression at a time. Use `&&` (“and”) to see if both expressions are true and `||` (“or”) to see if any one of the expressions is true.

```
if (first_expression_is_true && another_expression_is_true)
{
    // execute the appropriate commands when both expressions
    // are true
}
if (next_expression_is_true || last_expression_is_true)
{
    // execute the commands when at least one expression is
    // true
}
```

You can check for multiple independent (mutually exclusive) conditions and provide a default condition using `else if` and `else` options as follows:

```
if (first_expression_is_true)
{
    // execute the appropriate commands for this condition
}
```

```

else if (next_expression_is_true)
{
    // execute the commands related to this condition
}
else
{
    // this is the default if no expression above is true
}

```

You can use this logic to create a relatively robust `VendingMachine` class. Try to compile it as you go along. Create a new instance of the `VendingMachine` class and test all methods. Ensure your error checking is working correctly.

```

/**
 * VendingMachine.java
 * This class simulates a vending machine. Each machine can sell
 * one type of product at a fixed price.
 *
 * @author (Your name here)
 * @version (Today's date here)
 */
public class VendingMachine
{
    // instance variables
    private String productType; // type of product
    private int productPrice; // product's price
    private int numberSold; //total number of units sold
    private int balance; //amount of money buyer inserted
    private int totalMoney; //total $ taken by this machine
    //constructor
    /*
     * Default Constructor
     */
    public VendingMachine()
    {
        numberSold = 0;
        balance = 0;
        totalMoney = 0;
        productPrice = 0;
    }
    /*
     * Constructor with product Price and Type
     */
    public VendingMachine(String type, int price)
    {
        numberSold = 0;
        balance = 0;
        totalMoney = 0;
    }
}

```

```
        productType = type;
        productPrice = price;
    }
    //the public methods:
    /**
     * Method to set this machine's type of product
     *
     */
    public void setType(String type)
    {
        productType = type;
    } // end of setType()
    /**
     * Return this machine's type of product
     *
     */
    public String getType()
    {
        return productType;
    } // end of getType()
    /**
     * Allow buyer to insert money
     *
     */
    public void insertMoney(int money)
    {
        balance += money;
        totalMoney += money;
    } // end of insertMoney()
    /**
     * Allow user to set Price
     *
     */
    public void setPrice(int price)
    {
        productPrice = price;
    } // end of insertMoney()
    /**
     * Buy product
     *
     */
    public void buyProduct()
    {
        // Is there enough money in the machine
        // to buy this item
        double moreNeeded;
        if (balance < productPrice)
        {
            moreNeeded = productPrice - balance;
        }
    }
}
```

```

        System.out.println("You need to insert " +
            moreNeeded + " additional:");
    }
    else
    {
        giveProduct();
        giveChange();
    }
}
/**
 * Give the buyer the product
 *
 */
public void giveProduct()
{
    System.out.println("*-----*");
    System.out.println("| You bought |");
    System.out.println("| |");
    System.out.println("| " + productType);
    System.out.println("|-----|");
    numberSold++; // increase number sold by one
} // end of giveProduct()
/**
 * Give back change, if any, after a purchase
 *
 */
public void giveChange()
{
    balance -= productPrice; //how much to give back
    System.out.println("Your change is: " + balance);
    balance = 0; // clear balance
} // end of giveChange()
} // end of VendingMachine class

```

> Pacing/Timeline

- Code and debug the `VendingMachine` class with conditionals: 2–3 days
- Develop and test against a robust list of possible error conditions: 2–3 days

> Teacher Reflection (For example, what worked well in this lesson? What would you change if you were to teach it again?)

- Did students understand how to implement conditional statements?
- Did students recognize the role of conditional statements in validation?
- Did students have additional suggestions for using conditionals as a means of error checking?

> Addendum 1: Sample Code VendingMachine Class Without Error Checking

```

/**
 * VendingMachine.java
 * This class simulates a vending machine. Each machine can sell
 * one type of product at a fixed price.
 *
 * @author (Your name here)
 * @version (Today's date here)
 */

public class VendingMachine
{
    // instance variables
    private String productType; // type of product
    private int productPrice; // product's price
    private int numberSold; // total number of units sold
    private int balance; // amount of money buyer inserted
    private int totalMoney; // total $ taken by this machine
//constructor

//the public methods:
/**
 * Method to set this machine's type of product
 */
public void setType(String type)
{
    productType = type;
} // end of setType()

/**
 * Return this machine's type of product
 */
public String getType()
{
    return productType;
} // end of getType()

/**
 * Allow buyer to insert money
 */
public void insertMoney(int money)
{
    balance += money;
} // end of insertMoney()

/**
 * give the buyer the product
 */

```

```

public void giveProduct()
{
    System.out.println("*-----*");
    System.out.println("| You bought |");
    System.out.println("| |");
    System.out.println("| " + productType);
    System.out.println("|-----|");
    numberSold++; // increase number sold by one
} // end of giveProduct()

/**
 * Give the buyer's change back after giving product
 */
public void giveChange()
{
    balance -= productPrice; //how much to give back
    System.out.println("Your change is: " + balance);
    balance = 0; // clear balance
} // end of giveChange()

} // end of VendingMachine class

```

> Student Handout: Java Programming Error Checking With Conditionals

One problem with the basic `VendingMachine` class as illustrated in the addendum is that it lacks any error checking. That is, the user can enter a negative dollar amount or get a product before inserting enough money. This lesson will add error checking to prevent these and other error conditions from occurring. To do this, students will add conditional statements. The basic format of a conditional is as follows:



```

if (some_expression_is_true)
{
    // execute the appropriate commands
}

```

The method will execute the commands that exist between the brackets `{ }` only when the expression on the `if` line is true. When the expression is false, those commands will be ignored. You can extend conditionals in several ways. You can also check more than one expression at a time. Use `&&` (“and”) to see if both expressions are true and `||` (“or”) to see if any one of the expressions is true.



```

if (first_expression_is_true && another_expression_is_true)
{
    // execute the appropriate commands when both expressions are true}
}

```

```
    if (next_expression_is_true || last_expression_is_true)
    {
// execute the commands when at least one expression is true
    }
```

You can check for multiple independent (mutually exclusive) conditions and provide a default condition using `else if` and `else` options as follows:

```
if (first_expression_is_true)
{
    // execute the appropriate commands for this condition
}
else if (next_expression_is_true)
{
    // execute the commands related to this condition
}
...
else
{
    // this is the default if no expression above is true
}
```



Open the Vending project and the `VendingMachine` class in the editor and modify it as noted as follows. Try to compile your class as you go along. Create a new instance of the `VendingMachine` class and test all methods. Ensure your error checking is working correctly.

YOUR JOB:

- Enhance the `VendingMachine` class (see addendum) with conditionals to include error checking. Users should not be able to do the following:
 - Add a negative dollar amount
 - Add an amount exceeding some predetermined limit
 - Get a product until enough money has been inserted
 - Insert money if no products are available
- Comment everything
- Save, compile, and test OFTEN!

EXTRAS YOU CAN ADD:

- Get creative and make the program better. Add more error checking, offer more explicit error messages, and display more information for the user.