

# Table of Content

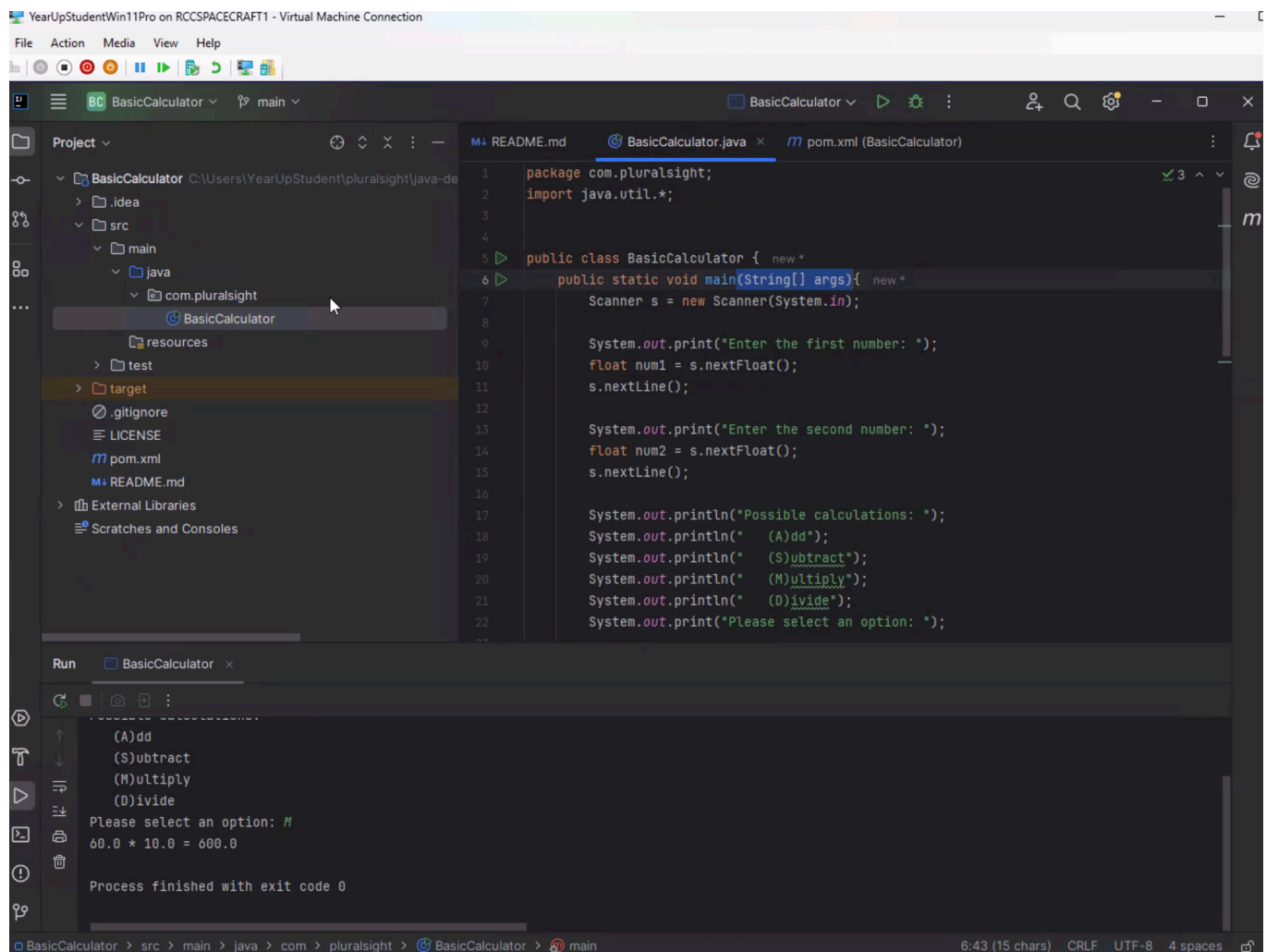
1. [Java Method](#)
2. [If/else if/ else](#)
3. [Exercise 4](#)
4. [Exercise 5](#)
5. [Exercise 6](#)

## Links

1. [Java Method Reference](#)
2. [Java IF-ELSE Reference](#)

# Notes

## Explain the Exercise Calculator.java



The screenshot displays an IDE window titled "YearUpStudentWin11Pro on RCCSPACECRAFT1 - Virtual Machine Connection". The main editor shows the `BasicCalculator.java` file with the following code:

```
1 package com.pluralsight;  
2 import java.util.*;  
3  
4  
5 public class BasicCalculator {  
6     public static void main(String[] args){  
7         Scanner s = new Scanner(System.in);  
8  
9         System.out.print("Enter the first number: ");  
10        float num1 = s.nextFloat();  
11        s.nextLine();  
12  
13        System.out.print("Enter the second number: ");  
14        float num2 = s.nextFloat();  
15        s.nextLine();  
16  
17        System.out.println("Possible calculations: ");  
18        System.out.println(" (A)dd");  
19        System.out.println(" (S)ubtract");  
20        System.out.println(" (M)ultiply");  
21        System.out.println(" (D)ivide");  
22        System.out.print("Please select an option: ");
```

The left sidebar shows the project structure for `BasicCalculator`, including `src/main/java/com/pluralsight/BasicCalculator`. The bottom panel shows the output of the program:

```
-----  
↑  
(A)dd  
↓  
(S)ubtract  
(M)ultiply  
(D)ivide  
Please select an option: M  
60.0 * 10.0 = 600.0  
Process finished with exit code 0
```

The status bar at the bottom indicates the file path: `BasicCalculator > src > main > java > com > pluralsight > BasicCalculator > main`, along with file statistics: 6:43 (15 chars), CRLF, UTF-8, 4 spaces.

# Java Method

In Java, a method is a block of code that performs a specific task. It is used to execute a particular operation, and it can take parameters, return a value, or both. Here's a breakdown of the components and structure of a Java method:

## Components of a Java Method

1. **Modifiers:** These define the access level and behavior of the method. Common modifiers include:
  - `public`: Accessible from other classes.
  - `private`: Accessible only within the same class.
  - `protected`: Accessible within the same package and subclasses.
  - `static`: Indicates that the method belongs to the class rather than instances of the class.
2. **Return Type:** The data type of the value returned by the method. Use `void` if no value is returned.
3. **Method Name:** A descriptive name that indicates what the method does, following Java naming conventions (camelCase).
4. **Parameters:** Inputs to the method, defined within parentheses. Multiple parameters are separated by commas.
5. **Method Body:** The block of code enclosed in braces `{}` that contains the instructions to be executed.

## Example of a Java Method

Here's an example of a simple Java method that calculates the sum of two integers:

```
public class Calculator {  
  
    // Method to calculate the sum of two integers  
    public int sum(int a, int b) {  
        int result = a + b; // Calculate the sum  
        return result; // Return the result  
    }  
  
    public static void main(String[] args) {  
        Calculator calculator = new Calculator(); // Create an instance of  
Calculator  
        int total = calculator.sum(5, 10); // Call the sum method  
    }  
}
```

```
        System.out.println("The sum is: " + total); // Output the result
    }
}
```

## Explanation of the Example

- **Class Declaration:** `public class Calculator` defines a class named `Calculator`.
- **Method Definition:**
  - `public int sum(int a, int b)` defines a method named `sum` that takes two integer parameters and returns an integer.
  - The method calculates the sum of `a` and `b` and returns it.
- **Main Method:**
  - `public static void main(String[] args)` is the entry point of the program.
  - An instance of `Calculator` is created to call the `sum` method, and the result is printed to the console.

## Calling a Method

You can call a method by using the following syntax:

```
objectName.methodName(arguments);
```

In the example above, the method `sum` is called using the instance of the `Calculator` class.

## Key Points

- Methods help to organize code and promote reusability.
- You can overload methods, which means defining multiple methods with the same name but different parameters.
- Methods can also be recursive, meaning they can call themselves.

---

## When to Use Methods

You use methods in programming when you want to:

1. **Encapsulate Logic:** To group related statements together that perform a specific task, making it easier to understand and manage your code.
2. **Reusability:** When you have code that you need to execute multiple times, methods allow you to write it once and call it whenever needed, avoiding code duplication.
3. **Organization:** To structure your code better, separating different functionalities or operations into distinct methods makes your codebase more organized and easier to navigate.
4. **Abstraction:** To hide complex implementation details from the user, providing a simple interface to work with (e.g., using library functions).
5. **Testing and Debugging:** To facilitate unit testing by isolating specific pieces of functionality, making it easier to identify and fix bugs.

## Why to Use Methods

1. **Improved Readability:** Methods can have descriptive names that clarify their purpose, making the code more understandable.
2. **Modularity:** By breaking down the code into smaller, manageable parts (methods), you create a modular program that is easier to maintain, modify, and extend.
3. **Reduction of Redundancy:** Methods help eliminate repeated code, which not only makes your program shorter and cleaner but also reduces the risk of errors.
4. **Ease of Maintenance:** When functionality is encapsulated in methods, changes can be made in one place without affecting other parts of the code, simplifying maintenance.
5. **Collaboration:** In team environments, methods allow different programmers to work on separate functionalities simultaneously, improving collaboration.

## Why Methods Exist

- **Structured Programming:** Methods embody the principles of structured programming by promoting the division of a program into smaller, manageable sections. This makes it easier to build complex software systems.
- **Efficiency:** By enabling code reuse and abstraction, methods contribute to more efficient programming practices, saving time and reducing the likelihood of errors.
- **Development Practices:** Modern software development relies on methods as a fundamental building block for creating clean, understandable, and maintainable code. This practice aligns with various software engineering principles, such as DRY (Don't Repeat Yourself) and KISS (Keep It Simple, Stupid).

# Summary

- **When to use:** To encapsulate logic, enhance reusability, organize code, provide abstraction, and aid testing.
- **Why to use:** For improved readability, modularity, reduction of redundancy, ease of maintenance, and collaboration.
- **Why it exists:** To promote structured programming, efficiency in development, and adherence to best practices in software engineering.

Methods are essential for writing clean, efficient, and maintainable code, enabling developers to tackle complex problems effectively.

---

In Java (and many other programming languages), methods can be categorized based on several criteria. Here are the main types of methods you'll encounter:

## 1. Based on Return Type

- **Void Method:** This type of method does not return a value. It performs an action but does not give any output to the caller.

**Example:**

```
public void printHello() {  
    System.out.println("Hello, World!");  
}
```

- **Return Method:** This type of method returns a value of a specific type. The return type is specified in the method declaration.

**Example:**

```
public int add(int a, int b) {  
    return a + b;  
}
```

## 2. Based on Parameters

- **No-Argument Method:** This method does not take any parameters.

**Example:**

```
public void greet() {  
    System.out.println("Hello!");  
}
```

- **Parameter Method:** This method accepts one or more parameters to perform its operations.

**Example:**

```
public void displayAge(int age) {  
    System.out.println("Age: " + age);  
}
```

### 3. Based on Scope

- **Instance Method:** These methods belong to an instance of a class and can access instance variables and methods. You need to create an object of the class to call an instance method.

**Example:**

```
public class Dog {  
    public void bark() {  
        System.out.println("Woof!");  
    }  
}  
  
Dog myDog = new Dog();  
myDog.bark(); // Calls the instance method
```

- **Static Method:** These methods belong to the class itself rather than any specific instance. They can be called without creating an object of the class. Static methods cannot access instance variables or methods directly.

**Example:**

```
public class MathUtils {  
    public static int square(int num) {  
        return num * num;  
    }  
}  
  
int result = MathUtils.square(5); // Calls the static method
```

## 4. Based on Functionality

- **Getter Method:** These methods are used to retrieve the value of an object's attribute. They usually return a value and do not take parameters.

**Example:**

```
public class Person {  
    private String name;  
  
    public String getName() {  
        return name;  
    }  
}
```

- **Setter Method:** These methods are used to set or update the value of an object's attribute. They usually take a parameter and do not return a value.

**Example:**

```
public class Person {  
    private String name;  
  
    public void setName(String name) {  
        this.name = name;  
    }  
}
```

## 5. Special Methods

- **Constructor Method:** These are special methods used to initialize new objects. Constructors have the same name as the class and do not have a return type.

**Example:**

```
public class Car {  
    private String model;  
  
    // Constructor  
    public Car(String model) {  
        this.model = model;  
    }  
}
```

- **Destructor Method:** In Java, destructors do not exist as in some other languages. However, you can use finalizers (though they are rarely used) to perform cleanup before an object is garbage collected.

## Summary

1. **Based on Return Type:**
  - Void Method
  - Return Method
2. **Based on Parameters:**
  - No-Argument Method
  - Parameter Method
3. **Based on Scope:**
  - Instance Method
  - Static Method
4. **Based on Functionality:**
  - Getter Method
  - Setter Method
5. **Special Methods:**
  - Constructor Method
  - Destructor Method (not commonly used in Java)

Understanding these types of methods helps in structuring your code efficiently, improving its readability, and ensuring it adheres to object-oriented programming principles.

---



# If/else if/ else

In Java, `if`, `else if`, and `else` are control flow statements that allow you to execute different blocks of code based on certain conditions. Here's a breakdown of how they work and some examples:

## Syntax

```
if (condition1) {  
    // code block to be executed if condition1 is true  
}  
else if (condition2) {  
    // code block to be executed if condition1 is false and condition2 is  
    true  
}  
else {  
    // code block to be executed if both condition1 and condition2 are false  
}
```

## Example

Here's a simple example to demonstrate the use of `if`, `else if`, and `else`:

```
public class Main {  
    public static void main(String[] args) {  
        int score = 85;  
  
        if (score >= 90) {  
            System.out.println("Grade: A");  
        } else if (score >= 80) {  
            System.out.println("Grade: B");  
        } else if (score >= 70) {  
            System.out.println("Grade: C");  
        } else if (score >= 60) {  
            System.out.println("Grade: D");  
        } else {  
            System.out.println("Grade: F");  
        }  
    }  
}
```

## Explanation

1. **If Statement:** Checks if `score` is greater than or equal to 90. If true, it prints "Grade: A".
2. **Else If Statement:** If the first condition is false, it checks if `score` is greater than or equal to 80. If true, it prints "Grade: B".
3. **Additional Else If Statements:** The process continues, checking lower thresholds (70, 60) for grades C and D.
4. **Else Statement:** If all previous conditions are false, it defaults to printing "Grade: F".

## Key Points

- You can have multiple `else if` conditions.
- The `else` block is optional and will execute only if none of the `if` or `else if` conditions are true.
- Only the first true condition's block will execute; the others will be skipped.

## Tips

- Make sure to use proper parentheses around conditions and curly braces for code blocks.
- Use meaningful variable names and comments to improve code readability.

---

**Inside the parentheses** `()` of an `if`, `else if`, or `else` statement in Java, you can place any valid boolean expression or condition. This expression evaluates to either `true` or `false`. Here are some common types of conditions you can use:

## 1. Comparison Operators

You can compare two values using comparison operators:

- **Equal to:** `==`
- **Not equal to:** `!=`
- **Greater than:** `>`
- **Less than:** `<`
- **Greater than or equal to:** `>=`
- **Less than or equal to:** `<=`

**Example:**

```
int a = 10;
int b = 20;

if (a < b) {
    System.out.println("a is less than b");
}
```

## 2. Logical Operators

You can combine multiple boolean expressions using logical operators:

- **AND:** `&&` (both conditions must be true)
- **OR:** `||` (at least one condition must be true)
- **NOT:** `!` (negates a boolean value)

**Example:**

```
boolean isRainy = true;
boolean isWeekend = false;

if (isRainy && isWeekend) {
    System.out.println("Stay indoors!");
} else if (isRainy || isWeekend) {
    System.out.println("You might want to take an umbrella.");
}
```

## 3. Method Calls

You can also call a method that returns a boolean value:

**Example:**

```
public static boolean isEven(int number) {
    return number % 2 == 0;
}

public static void main(String[] args) {
    int num = 4;

    if (isEven(num)) {
        System.out.println(num + " is even.");
    }
}
```

```
    } else {  
        System.out.println(num + " is odd.");  
    }  
}
```

## 4. Boolean Variables

You can directly use boolean variables:

**Example:**

```
boolean isLoggedIn = false;  
  
if (isLoggedIn) {  
    System.out.println("Welcome back!");  
} else {  
    System.out.println("Please log in.");  
}
```

## 5. Complex Conditions

You can combine various conditions to create complex expressions:

**Example:**

```
int age = 25;  
boolean hasLicense = true;  
  
if (age >= 18 && hasLicense) {  
    System.out.println("You can drive.");  
} else {  
    System.out.println("You cannot drive.");  
}
```

## Summary

You can put any expression that evaluates to a boolean (true or false) inside the parentheses of `if`, `else if`, or `else` statements. This gives you flexibility in controlling the flow of your program based on different conditions.

---

## When to Use `if`, `else if`, and `else`

You use `if`, `else if`, and `else` statements when you want your program to make decisions based on certain conditions. Here are some common scenarios:

1. **User Input Validation:** Checking if the input from the user meets certain criteria (e.g., age, format, etc.).
2. **Conditional Logic:** Executing different blocks of code depending on the state of your program (e.g., checking if a user is logged in).
3. **Control Flow:** Directing the flow of execution based on conditions, such as in games or applications where different outcomes are possible based on user actions.
4. **Error Handling:** Implementing responses to unexpected conditions (e.g., handling exceptions, validating data).

## Why to Use `if`, `else if`, and `else`

1. **Decision Making:** These constructs allow you to implement logic that makes your program more dynamic. They enable your code to react differently based on varying conditions.
2. **Improved Readability:** Using conditional statements can help make the intent of your code clear. By structuring your code with these statements, it can be easier for others (or yourself) to understand the logic at a glance.
3. **Control Flow:** They allow you to control the flow of your program, executing different code paths based on specific criteria. This is essential for creating interactive and user-driven applications.
4. **Flexibility:** You can easily modify the conditions and responses in your code without altering the overall structure, making your code adaptable to changes.

## Why It Exists

- **Programming Paradigm:** Conditional statements are fundamental to programming languages because they enable logical reasoning within the code. They represent a basic concept of control flow that allows for more complex operations and decision-making.
- **Human Logic:** Conditional statements reflect human reasoning. Just as we evaluate situations and make decisions based on conditions (e.g., "If it rains, I'll take an umbrella"), programming needs to mimic this decision-making process.
- **Algorithm Implementation:** Many algorithms rely on conditions to operate correctly, whether it's sorting data, searching for values, or making calculations based on criteria.

## Summary

- **When to use:** In situations requiring decisions based on conditions (user input, control flow, etc.).
- **Why to use:** For decision-making, improved readability, control flow, and flexibility.
- **Why it exists:** To reflect human logic in programming, enable complex algorithm implementation, and provide fundamental control over program execution.

Conditional statements are essential tools that make programming powerful and versatile, allowing developers to create dynamic and responsive applications.

---

## Exercise 4

```
package com.pluralsight;

import java.util.Scanner;

/**
 * The PayrollCalculator class provides a simple payroll calculator * that
 * prompts the user for their name, work hours, and work rate, then calculates
 * and displays their salary. */public class PayrollCalculator {

    // Scanner object to read user input
    static Scanner scanner = new Scanner(System.in);

    /**
     * The main method is the entry point of the application.      * It
     * prompts the user for their name, work hours, and work rate,      * then
     * calculates and displays their salary.      *      * @param args Command line
     * arguments (not used)
     */
    public static void main(String[] args) {
        System.out.println("Enter your name: ");
        String name = scanner.nextLine();

        System.out.println("Enter your work hours: ");
        float hours = scanner.nextFloat();

        System.out.println("Enter your work rate: ");
        float rate = scanner.nextFloat();

        float salary = 0;
        if (hours < 40) {
```

```

        salary = rate * hours;
    } else {
        float overtimeRate = rate * 1.5f;
        System.out.println("overtimeRate: " + overtimeRate);
        float overtimeHours = hours - 40;
        System.out.println("overtimeHours: " + overtimeHours);
        float overSalary = overtimeRate * overtimeHours;
        System.out.println("overSalary: " + overSalary);
        float regularSalary = rate * 40f;
        System.out.println("regularSalary: " + regularSalary);
        salary = overSalary + regularSalary;
    }

    System.out.println(name + "'s salary is " + salary);
}
}

```

## Exercise 5

```

package com.pluralsight;

import java.util.Scanner;

/**
 * The PayrollCalculatorWithMethod class provides a simple payroll
 * calculator * that prompts the user for their name, work hours, and work
 * rate, then calculates * and displays their salary. */public class
PayrollCalculatorWithMethod {

    // Scanner object to read user input
    static Scanner scanner = new Scanner(System.in);

    /**
     * The main method is the entry point of the application.      * It
     * prompts the user for their name, work hours, and work rate,      * then
     * calculates and displays their salary.      *      * @param args Command line
     * arguments (not used)
     */
    public static void main(String[] args) {
        String name = promptForName();
        float hours = promptForFloat("hours");
        float rate = promptForFloat("rate");
        float salary = payrollCalculator(hours, rate);
        System.out.println(name + "'s salary is " + salary);
    }
}

```

```

/**
 * Prompts the user for a float value based on the given option.
 * @param option The type of input to prompt for ("hours" or "rate")
 * @return The float value entered by the user
 */
private static float promptForFloat(String option) {
    float nextFloat = 0f;
    if (option.equalsIgnoreCase("hours")) {
        System.out.println("Enter your work hours: ");
    } else if (option.equalsIgnoreCase("rate")) {
        System.out.println("Enter your work rate: ");
    }

    try {
        nextFloat = scanner.nextFloat();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return nextFloat;
}

/**
 * Prompts the user for their name.
 * @return The name entered by the user
 */
private static String promptForName() {
    String name = "";
    try {
        System.out.println("Enter your name: ");
        name = scanner.nextLine();
    } catch (Exception e) {
        e.printStackTrace();
    }

    return name;
}

/**
 * Calculates the salary based on the given hours and rate.
 * If hours are less than 40, the salary is calculated as rate * hours.
 * If hours are 40 or more, the salary includes overtime pay.
 * @param hours The number of hours worked
 * @param rate The hourly rate
 * @return The calculated salary
 */
private static float payrollCalculator(float hours, float rate) {

```



```

    float salary = 0;

    if (hours < 40) {
        salary = rate * hours;
    } else {
        float overtimeRate = rate * 1.5f;
        float overtimeHours = hours - 40;
        float overSalary = overtimeRate * overtimeHours;
        float regularSalary = rate * 40f;
        salary = overSalary + regularSalary;
    }

    return salary;
}
}

```

## Exercise 6

```

package com.pluralsight;

import java.time.LocalDate;
import java.time.format.DateTimeFormatter;
import java.util.Scanner;

/**
 * This class calculates the rental cost for a car based on various
 * parameters such as rental days, age, and additional features. */public class
RentalCarCaculator {

    static Scanner scanner = new Scanner(System.in);
    final static double BASIC_CAR_RENTAL_PRICE = 29.99;
    final static double UNDER_AGE_CHARGE = 0.3;
    final static double TOLL_TAG_CHARGE = 3.95;
    final static double GPS_CHARGE = 2.95;
    final static double SOS_CHARGE = 3.95;

    /**
     * The main method to run the rental car calculator.      * @param args
     * Command line arguments.
     */    public static void main(String[] args) {
        String pickUpDate = promptForPickUpDate();
        int numberOfDayRental = promptForRentalDays();
        boolean tollTag = promptForFeature("TOLL");
        boolean GPS = promptForFeature("GPS");
    }
}

```

```

        boolean SOS = promptForFeature("SOS");
        int age = promptForAge();

        calculator(pickUpDate, numberOfDayRental, tollTag, GPS, SOS, age);
    }

    /**
     * Calculates the total rental cost based on the provided parameters.
     * @param pickUpDate The pick-up date in MM-dd-yyyy format.
     * @param numberOfDayRental The number of days the car is rented.
     * @param tollTag Whether the toll tag feature is selected.
     * @param gps Whether the GPS feature is selected.
     * @param sos Whether the SOS feature is selected.
     * @param age The age of the renter.
     */
    private static void calculator(String pickUpDate, int
numberOfDayRental, boolean tollTag, boolean gps, boolean sos, int age) {
        DateTimeFormatter formatter = DateTimeFormatter.ofPattern("MM-dd-
yyyy");
        double total = 0;

        try {
            LocalDate date = LocalDate.parse(pickUpDate, formatter);
            System.out.println(date);

            if (tollTag) {
                total += numberOfDayRental * TOLL_TAG_CHARGE;
            }

            if (gps) {
                total += numberOfDayRental * GPS_CHARGE;
            }

            if (sos) {
                total += numberOfDayRental * SOS_CHARGE;
            }

            if (age > 25) {
                total += numberOfDayRental * BASIC_CAR_RENTAL_PRICE;
            } else {
                total += (numberOfDayRental * BASIC_CAR_RENTAL_PRICE) +
(numberOfDayRental * BASIC_CAR_RENTAL_PRICE * UNDER_AGE_CHARGE);
            }

            System.out.println("Your start day will be: " + pickUpDate);
            System.out.println("Your total cost for renting " +
numberOfDayRental + " days is " + total);

```

```

        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }

    /**
     * Prompts the user to enter their age.      * @return The age of the
user.
    */
    private static int promptForAge() {
        System.out.println("Enter your age: ");
        int age = 0;
        try {
            age = scanner.nextInt();
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
        return age;
    }

    /**
     * Prompts the user to select a feature.      * @param option The feature
option (TOLL, GPS, SOS).
     * @return True if the feature is selected, false otherwise.
    */
    private static boolean promptForFeature(String option) {
        try {
            String flag = scanner.nextLine();

            if (option.toLowerCase().trim().equals("sos")) {
                System.out.println("Would you like the roadside assistant?
for ($3.95/day) Yes or No");
                scanner.nextLine();
            } else if (option.toLowerCase().trim().equals("gps")) {
                System.out.println("Would you like the GPS feature?  for
($2.95/day) Yes or No");
            } else if (option.toLowerCase().trim().equals("toll")) {
                System.out.println("Would you like the E-Toll Tag?  for
($3.95/day) Yes or No");
            } else {
                System.out.println("No features available");
            }
            if (flag.trim().toLowerCase().equals("yes")) {
                return true;
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

```

```

        return false;
    }

    /**
     * Prompts the user to enter the number of rental days.      * @return
The number of rental days.
    */
    private static int promptForRentalDays() {
        System.out.println("How many days would you like to rent? : ");
        int rentDays = 0;
        try {
            rentDays = scanner.nextInt();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return rentDays;
    }

    /**
     * Prompts the user to enter the pick-up date.      * @return The pick-up
date in MM/dd/yy format.
    */
    private static String promptForPickUpDate() {
        System.out.println("Enter your pick up day (mm/dd/yy): ");
        String date = null;
        try {
            date = scanner.nextLine();
        } catch (Exception e) {
            e.printStackTrace();
        }
        return date;
    }
}

```