Questions / Answers

Mid Sem

What is the use of String class over character array? Write a program to find maximum length palindrome substring from a given string

- String immutable (explain), therefore thread safe
- in-built methods- substring, length, replace.
- has hashcode(), therefore can use as keys on HashMap.

```
String b = "babaddab";
int size = b.length();
for (int i = 0; i < b.length(); i++) {
    for (int j = 0; j + size <= b.length(); j++) {
        String check = b.substring(j, j + size);
        String rev = (new StringBuffer(check).reverse()).toString();
        if (check.equals(rev)) {
            System.out.println(check);
            break;
        }
    }
    size--;
}
```

What is the use of package in Java ? Write a program to show the working of protected keyword

Packages in Java provide a way to organize related classes and interfaces together. It helps in avoiding naming conflicts and provides better code maintainability. Packages in Java can be imported using the import statement to make the classes and interfaces of a package accessible in another package.

The protected keyword is an access modifier in Java which allows the members of a class to be accessed within the same package or by a subclass in a different package. Here is an example program that shows the working of the protected keyword:

```
package com.example;

public class Superclass {
    protected int protectedVariable = 10;
}

package com.example;

public class Subclass extends Superclass {
    public void display() {
        System.out.println("Protected variable value: " + protectedVariable);
    }
}

package com.example;

public class Main {
    public static void main(String[] args) {
        Subclass obj = new Subclass();
        obj.display();
    }
}
```

In the above program, we have a superclass Superclass with a protected member variable protectedVariable. We then have a subclass Subclass which extends Superclass and has a public method display that accesses the protectedVariable. Finally, we have a Main class that creates an object of Subclass and calls its display method.

Since protectedVariable is declared as protected, it is accessible within the same package (com.example) and also by the subclass Subclass which is in a different package but extends Superclass.

When we run the Main class, the output will be:

```
Protected variable value: 10
```

This shows that the protected keyword has allowed the Subclass to access the protectedVariable of its superclass.

What is method overiding? Write a program to show the use of this keyword properly while overriding any of methods of object class

Method overriding in Java is the process of defining a method in a subclass that has the same name, return type, and parameters as a method in its superclass. The method in the subclass overrides the method in the superclass, and the behavior of the method in the subclass is determined by the implementation in the subclass.

The `this` keyword in Java is a reference to the current object. It can be used to refer to instance variables and methods of the current object. When a method in a subclass overrides a method in its superclass, we can use the `super` keyword to call the overridden method in the superclass, and the `this` keyword to call the method in the current object.

Here is an example program that demonstrates the use of `this` keyword while overriding a method of the `Object` class:

```
public class MyClass {
    private int id;

    public MyClass(int id) {
        this.id = id;
    }
    @Override
    public boolean equals(Object obj) {
        if (this == obj) {
            return true;
        }
        if (!(obj instanceof MyClass)) {
            return false;
        }
        MyClass other = (MyClass) obj;
        return this.id == other.id;
    }
}
```

In the above program, we have a class `MyClass` with a private member variable `id` and a constructor that initializes it. We have overridden the `equals` method of the `Object` class to compare the `id` member variables of two `MyClass` objects for equality.

We have used the `this` keyword in the `equals` method to refer to the `id` member variable of the current object, and the `super` keyword to call the `equals` method of the `Object` class. The `equals` method of the `Object` class takes an `Object` as its parameter, so we need to cast it to a `MyClass` object to access its `id` member variable.

By using the `this` keyword and the `super` keyword properly, we have overridden the `equals` method of the `Object` class in a way that compares the `id` member variables of two `MyClass` objects for equality.

Write any one disadvantage of using static import in Java. Write program to show the use of super keyword in three-way.

Disadvantage of using static import in Java:

- It can make code harder to read and understand, especially for other developers who are not familiar with the static methods or constants being imported.

Example program demonstrating the use of `super` keyword in three ways:

```
public class Animal {
    String name = "Animal";

    public void printName() {
        System.out.println("Name: " + name);
    }
}

public class Dog extends Animal {
    String name = "Dog";

    @Override
    public void printName() {
        // Calling the printName() method of the parent class using super
        super.printName();
        System.out.println("Name: " + name);
    }

    public void printSuperName() {
        // Accessing the name variable of the parent class using super
        System.out.println("Super Name: " + super.name);
    }

    public void callSuperConstructor() {
        // Calling the constructor of the parent class using super
        super();
    }
}

public class Main {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.printName(); // Output: Name: Animal \n Name: Dog
        myDog.printSuperName(); // Output: Super Name: Animal
        myDog.callSuperConstructor(); // This will call the constructor of the
parent class
    }
}
```

In the above example, the `super` keyword is used in three different ways:

- In the `printName()` method of the `Dog` class, `super.printName()` is used to call the `printName()` method of the parent class, which prints the value of the `name` variable of the parent class.
- In the `printSuperName()` method of the `Dog` class, `super.name` is used to access the `name` variable of the parent class.
- In the `callSuperConstructor()` method of the `Dog` class, `super()` is used to call the constructor of the parent class.

What is the use of command line arguments ? Write a program to print Sum or all command line arguments which are positive integers.

Sure, here's an example program in Java that prints the sum of all command line arguments that are positive integers:

```
public class PositiveIntegers {
    public static void main(String[] args) {
        int sum = 0;
        for (String arg : args) {
            try {
                int n = Integer.parseInt(arg);
                if (n > 0) {
                    sum += n;
                }
            } catch (NumberFormatException e) {
                // Ignore non-integer arguments
            }
        }
        System.out.println("Sum of positive integers: " + sum);
    }
}
```

Here's how the program works:

- It defines a class called `PositiveIntegers`.
- It defines a `main` method, which takes an array of strings as an argument.
- It initializes a variable `sum` to zero.
- It loops through each element of the `args` array, which contains the command line arguments.
- For each argument, it attempts to parse it as an integer using `Integer.parseInt`.
- If the parsing succeeds and the resulting integer is positive, it adds it to the `sum`.
- If the parsing fails (because the argument is not an integer), it ignores that argument and moves on to the next one.
- After processing all arguments, it prints the sum of positive integers.

To run this program, you can open a command prompt or terminal, navigate to the directory containing the compiled `.class` file, and enter a command like this:

```
PositiveIntegers 1 2 3 -4 5 foo 6 bar
```

This will run the program and pass it several command line arguments. The output should be:

```
Sum of positive integers: 17
```

Note that the program only counts positive integers (1, 2, 3, 5, 6) and ignores non-integer arguments (-4, "foo", "bar").

What is array of objects in Java ? Write a program to initialize integer array of size ten at runtime. Print sum of all elements which are having first starting as even value.

Array of objects in Java refers to creating an array where each element is an object of a class. It allows us to create multiple objects of a class and store them in an array.

Here is an example program that initializes an integer array of size ten at runtime and prints the sum of all elements which have the first digit as an even value:

```
import java.util.Scanner;

public class ArrayOfObjectsExample {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        Integer[] arr = new Integer[10]; // creating integer array of size ten

        // initializing the array elements at runtime
        System.out.println("Enter 10 integer values:");
        for(int i = 0; i < 10; i++) {
            arr[i] = sc.nextInt();
        }

        int sum = 0;
        for(int i = 0; i < 10; i++) {
            // get the first digit of the current element
            int firstDigit = Math.abs(arr[i]); // taking absolute value to handle
negative numbers
            while(firstDigit >= 10) {
                firstDigit = firstDigit / 10;
            }

            // check if the first digit is even
            if(firstDigit % 2 == 0) {
                sum += arr[i]; // add the element to the sum
            }
        }

        System.out.println("Sum of elements with even first digit: " + sum);
    }
}
```

In the above program, we create an array of `Integer` objects of size 10 using the `Integer[] arr = new Integer[10]` syntax. We then use a `Scanner` object to initialize the elements of the array at runtime.

We then iterate over the array and check if the first digit of each element is even. If it is, we add the element to the `sum` variable. Finally, we print the value of `sum`, which is the sum of all elements that have an even first digit.

What is the use of final and finally keyword in Java ? Write all steps to Create an executable jar file. How can we use a package class that is part of some jar file?

The final keyword in Java is used to restrict the user from modifying the value of a variable, method or class. Once a final keyword is used, the value cannot be changed throughout the program.

The finally keyword in Java is used to define a block of code that will be executed after a try-catch block. The finally block is used to ensure that a specific piece of code is executed, even if an exception is thrown. It is usually used to release resources that were acquired in the try block.

Steps to create an executable jar file:

1. Compile all the Java files that you want to include in the jar file using the command: javac .java
2. Create a manifest file that contains the main class name and the classpath. The manifest file should be named MANIFEST.MF and should be placed in the META-INF folder.
3. Create the jar file using the command: jar cvfm .jar META-INF/MANIFEST.MF
4. Test the jar file by running the command: java -jar .jar

To use a package class that is part of some jar file, follow these steps:

1. Add the jar file to your classpath using the command: java -cp .jar
2. Import the package in your Java program using the import statement: import .;
3. Use the class in your Java program.

What is difference between interface and abstract class ? Write a program to show how we can use an interface in upcasting.

The main differences between an interface and an abstract class in Java are:

- An interface is a pure abstraction, where all the methods declared in an interface are abstract and do not have any implementation. On the other hand, an abstract class can have both abstract and non-abstract methods.
- A class can implement multiple interfaces, but it can extend only one abstract class.
- An interface cannot have any instance variables, while an abstract class can have instance variables.

Here's an example program to show how we can use an interface in upcasting:

```java
interface Vehicle {
    void drive();
}

class Car implements Vehicle {
    @Override
    public void drive() {
        System.out.println("Driving a car...");
    }

    public void park() {
        System.out.println("Parking the car...");
    }
}

public class Main {
    public static void main(String[] args) {
        Vehicle myVehicle = new Car();
        myVehicle.drive();
    }
}
```

In the above example, we define an interface called `Vehicle` that has a method called `drive()`. We then create a class called `Car` that implements the `Vehicle` interface and overrides the `drive()` method. The `Car` class also has an additional method called `park()`.

In the `main()` method, we create an object of `Car` class and assign it to a reference of `Vehicle` interface. This is known as upcasting. We then call the `drive()` method on the `myVehicle` object. Since `Car` class implements the `Vehicle` interface and overrides the `drive()` method, it is able to provide its own implementation of the `drive()` method when it is called on the `myVehicle` object.

What is the use of StringBuffer over String Class ? Write a program to enter a string and print all characters removing duplicates from it.

The `StringBuffer` class in Java is used to create mutable strings. It provides various methods to modify the string content such as `append()`, `insert()`, `replace()`, `delete()`, etc. On the other hand, the `String` class creates immutable strings that cannot be modified once created.

Here is a program to enter a string and print all characters removing duplicates from it using `StringBuffer`:

```
import java.util.Scanner;

public class RemoveDuplicates {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String input = scanner.nextLine();

        StringBuffer result = new StringBuffer();
        for (int i = 0; i < input.length(); i++) {
            char c = input.charAt(i);
            if (result.indexOf(String.valueOf(c)) == -1) {
                result.append(c);
            }
        }
        System.out.println("String after removing duplicates: " +
result.toString());
    }
}
```

In the above program, we first create a `Scanner` object to read user input. Then we prompt the user to enter a string and read it using the `nextLine()` method.

Next, we create a `StringBuffer` object named `result`. We then loop through each character of the input string using a `for` loop. For each character, we check if it already exists in the `result` string using the `indexOf()` method. If the character is not present in the `result` string, we append it to the `result` string using the `append()` method.

Finally, we print the `result` string after removing duplicates using the `toString()` method.

Note that we could also achieve the same result using the `HashSet` class.

What are checked and unchecked exceptions in Java ? Write a program to generate and throw custom exception name MyException if there is no integer digit in any user entered String value.

Checked and unchecked exceptions are two types of exceptions in Java. Checked exceptions are those that are checked by the compiler at compile-time, and the developer is required to handle them using try-catch or throw them using the throws keyword. Examples of checked exceptions include IOException, SQLException, etc. On the other hand, unchecked exceptions are those that are not checked by the compiler at compile-time, and the developer is not required to handle them explicitly. Examples of unchecked exceptions include NullPointerException, ArrayIndexOutOfBoundsException, etc.

Here is an example program that generates and throws a custom exception named MyException if there is no integer digit in any user-entered String value:

```
import java.util.Scanner;

public class CustomExceptionExample {
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String inputString = scanner.nextLine();
        scanner.close();

        try {
            int num = Integer.parseInt(inputString);
            System.out.println("You entered: " + num);
        } catch (NumberFormatException e) {
            throw new MyException("No integer found in input string");
        }
    }
}

class MyException extends RuntimeException {
    public MyException(String message) {
        super(message);
    }
}
```

In this program, the user is prompted to enter a string. The program then tries to convert this string into an integer using the `Integer.parseInt()` method. If the conversion is successful, the program prints the integer value. However, if the conversion fails due to the absence of any integer digit in the input string, a custom exception named MyException is thrown with an appropriate error message. Note that `MyException` extends `RuntimeException`, making it an unchecked exception that does not need to be handled explicitly.

End stuff

How the working of String is diferent from StringBuffer ? Write a program to print total numbers of vowels using switch case in an entered string.

The main difference between String and StringBuffer in Java is that String is immutable, meaning its value cannot be changed once it is created, while StringBuffer is mutable, meaning its value can be changed after it is created.

Here's an example program to count the total number of vowels in a string using switch case:

```
import java.util.Scanner;
```

```java
public class VowelCount {
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.print("Enter a string: ");
        String str = sc.nextLine();
        sc.close();

        int vowelCount = 0;

        for(int i=0; i<str.length(); i++) {
            char ch = str.charAt(i);
            switch(ch) {
                case 'a':
                case 'e':
                case 'i':
                case 'o':
                case 'u':
                case 'A':
                case 'E':
                case 'I':
                case 'O':
                case 'U':
                    vowelCount++;
                    break;
                default:
                    // do nothing
            }
        }

        System.out.println("Total number of vowels in the string: " + vowelCount);
    }
}
```

In this program, we take a string input from the user and count the number of vowels in the string using a `for` loop and a `switch` statement. We check each character of the string and increment the `vowelCount` variable if the character is a vowel. Finally, we print the total number of vowels in the string.

What are the use of Wrapper classes? Write a program to show the use of command line arguments in order to check whether any of command line arguments is 100 or not.

Sure, here's an example program to demonstrate the use of Wrapper classes and command line arguments in Java:

```java
public class CommandLineArgsDemo {
    public static void main(String[] args) {
        // check if any of the command line arguments is equal to 100
        for (String arg : args) {
            try {
                if (Integer.parseInt(arg) == 100) {
                    System.out.println("100 is present as a command line
argument.");
```

```
                    return;
                }
            } catch (NumberFormatException e) {
                // ignore non-integer command line arguments
            }
        }
        System.out.println("100 is not present as a command line argument.");
    }
}
```

In this program, we iterate over the `args` array to check if any of the command line arguments is equal to 100. We use the `Integer.parseInt()` method to convert each argument to an `int` value, and then compare it with 100. Since `Integer.parseInt()` throws a `NumberFormatException` if the argument is not a valid integer, we catch this exception and ignore non-integer command line arguments.

Note that we use the `Integer` wrapper class to convert a `String` value to an `int` value using the `parseInt()` method. This is an example of the use of Wrapper classes, which are used to convert between primitive types and their corresponding object types. In this case, we convert the `String` representation of an integer to an `int` value using the `Integer` wrapper class.

What are multidimensional arrays in java? Write a program to initlize 3 by 3 integer array and print sum of both diagonal elements.

Sure, here's an example program to initialize a 3 by 3 integer array and print the sum of both diagonal elements:

```
public class MultiDimensionalArrayExample {
  public static void main(String[] args) {
    int[][] arr = {{1,2,3}, {4,5,6}, {7,8,9}};
    int sum = 0;
    for (int i = 0; i < arr.length; i++) {
      for (int j = 0; j < arr[i].length; j++) {
        if (i == j || i + j == arr.length - 1) {
          sum += arr[i][j];
        }
      }
    }
    System.out.println("Sum of both diagonal elements: " + sum);
  }
}
```

In this program, we first initialize a 3 by 3 integer array using curly braces notation. We then use nested loops to iterate over each element of the array. For each element, we check if it is on either diagonal (i.e. the row index is equal to the column index, or the row index plus the column index equals the length of the array minus 1). If it is on either diagonal, we add the element to the running sum. Finally, we print the sum of both diagonal elements.

What are checked and unchecked exceptions? Write a program to show the use of try with resource properly.

In Java, exceptions are categorized into two types: checked exceptions and unchecked exceptions.

Checked exceptions are the exceptions that are checked at compile time by the compiler. These exceptions must either be caught or declared in the method signature using the 'throws' keyword. Some examples of checked exceptions in Java are IOException, ClassNotFoundException, and SQLException.

Unchecked exceptions, on the other hand, are not checked at compile time by the compiler. These exceptions are usually caused by errors in the program logic or unexpected conditions during runtime. Some examples of unchecked exceptions in Java are NullPointerException, ArrayIndexOutOfBoundsException, and ArithmeticException.

Here's an example program that demonstrates the use of try-with-resources statement:

```
import java.io.*;

public class Example {
  public static void main(String[] args) {
    try (FileReader fileReader = new FileReader("file.txt");
         BufferedReader bufferedReader = new BufferedReader(fileReader)) {

      String line;
      while ((line = bufferedReader.readLine()) != null) {
        System.out.println(line);
      }

    } catch (IOException e) {
      System.out.println("An error occurred: " + e.getMessage());
    }
  }
}
```

In the above program, we are reading a file named "file.txt" using FileReader and BufferedReader. Instead of using try-catch-finally block to handle the resources, we are using the try-with-resources statement. This statement automatically closes the resources that are declared inside the parentheses after the try block finishes executing. It ensures that the resources are closed properly, even in the case of an exception being thrown. This way, we don't have to manually close the resources using the 'finally' block, making the code more concise and less prone to errors.

How interfaces are used in multiple inheritance? Write a program to differentiate inner class and anonymous inner class.

In Java, interfaces are used to achieve multiple inheritance as a class can implement multiple interfaces. When a class implements multiple interfaces, it inherits the abstract methods from all of the interfaces and must provide the implementation for each of them.

Here's an example program to differentiate between inner class and anonymous inner class:

```java
public class InnerClassExample {

  private int x = 10;

  // Inner class
  public class MyInnerClass {
    public void display() {
      System.out.println("Value of x from inner class: " + x);
    }
  }

  public static void main(String[] args) {
    InnerClassExample outer = new InnerClassExample();

    // Accessing inner class
    InnerClassExample.MyInnerClass inner = outer.new MyInnerClass();
    inner.display();

    // Anonymous inner class
    Runnable r = new Runnable() {
      public void run() {
        System.out.println("Value of x from anonymous inner class: " + outer.x);
      }
    };

    // Starting thread using anonymous inner class
    Thread t = new Thread(r);
    t.start();
  }
}
```

In the above program, `MyInnerClass` is an inner class defined within the `InnerClassExample` class. We can access it using the `outer.new MyInnerClass()` syntax. On the other hand, an anonymous inner class is defined without a name using the `new` keyword and can be used to implement an interface or extend a class. In the example, we have created an anonymous inner class to implement the `Runnable` interface and start a new thread.