Q1.Write a simple Banking System program by using OOPs concept where you can get account Holder n
ame balance etc?

Ans

```
package in.ineuron.bank;

import java.util.Scanner;

public class BankingApp {
 public static String bankName = "A1BANK";
 public String name;
 public double balance;

 BankingApp(String name)

 {
  this.name = name;
 }

 public void deposit(double amount) {
  balance = balance + amount;
  System.out.println("After Deposited The remaining Balance is " + balance);
 }

 public void withdraw(double amount) {
  if (amount > balance) {
   System.out.println("Insufficient Fund ....can’t perform this operation");
   System.exit(0);
  } else {
   balance = balance - amount;
   System.out.println("After withdraw The remaining Balance is " + balance);

  }
 }

 public static void main(String[] args) {
  System.out.print("Welcome to Java" + BankingApp.bankName);
  Scanner sc = new Scanner(System.in);
  System.out.println("Enter Your Name");
  String name = sc.next();
  BankingApp b1 = new BankingApp(name);
  System.out.println("*Congrats " + b1.name + " Your Account Got Created* ");
  while (true) {
   System.out.println("Choose Your to perform:");
   System.out.println("D-Deposit \nW-WithDraw \nE-Exit \nC-Check Available Balance");
   String option = sc.next();
   if (option.equalsIgnoreCase("D")) {
    System.out.println("Enter amount to Deposit......");
    double amount = sc.nextDouble();
    b1.deposit(amount);
   } else if (option.equalsIgnoreCase("W")) {
    System.out.println("Enter amount to withdraw......");
    double amount = sc.nextDouble();
    b1.withdraw(amount);
   } else if (option.equalsIgnoreCase("c")) {
    System.out.println("Your Available Account Balance is  " + b1.balance);

   } else if (option.equalsIgnoreCase("E")) {
    System.out.println("Thanks For Banking with Us......");
    System.exit(0);
   } else {
    System.out.println("Invalid Entry......\nPlease Choose Valid Options...... ");
   }
  }

 }

}
```

Q2. Write a Program where you inherit method from parent class and show method Overridden Concept?

Ans

```
class Animal {
    public void speak() {
        System.out.println("Animals speak!");
    }
}

class Dog extends Animal {
    @Override
    public void speak() {
        System.out.println("Dogs bark!");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal animal = new Animal();
        animal.speak();

        Dog dog = new Dog();
        dog.speak();
    }
}
```

Q3.Write a program to show run time polymorphism in java?

Ans

```
class Shape {
    public void draw() {
        System.out.println("Drawing...");
    }
}
```

```
class Rectangle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a rectangle...");
    }
}

class Circle extends Shape {
    @Override
    public void draw() {
        System.out.println("Drawing a circle...");
    }
}

public class Main {
    public static void main(String[] args) {
        Shape shape = new Rectangle();
        shape.draw();

        shape = new Circle();
        shape.draw();
    }
}
```

Q4.Write a program to show Compile time polymorphism in java?

Ans

```
class Overloading {
    public void sum(int a, int b) {
        System.out.println(a + b);
    }

    public void sum(int a, int b, int c) {
        System.out.println(a + b + c);
    }

    public static void main(String[] args) {
        Overloading overloading = new Overloading();
        overloading.sum(1, 2);
        overloading.sum(1, 2, 3);
    }
}
```

Q5. Achieve loose coupling in java by using OOPs  concept?

Ans

```
public interface Topic
{
    void understand();
}
class Topic1 implements Topic {
```

```
public void understand()
   {
      System.out.println("Got it");
   }
} class Topic2 implements Topic {
public void understand()
   {
      System.out.println("understand");
   }
} public class Subject {
public static void main(String[] args)
   {
      Topic t = new Topic1();
      t.understand();
   }
}
```

Q6. What is the benefit of encapsulation in java?

Ans

Encapsulation is the process of wrapping up data and the code that operates on that data within a single unit, which is called a class in Java. This helps to keep the data and code safe from outside interference.

There are several benefits to encapsulation in Java:

Data hiding: Encapsulation allows you to hide the data within a class from other classes. This can help to protect the data from being modified or accessed by unauthorized code.

Code reuse: Encapsulation can help you to reuse code by creating reusable classes. This can make your code more modular and easier to maintain.

Testability: Encapsulation can make your code more testable by isolating the data and code within a class . This can make it easier to write unit tests for your code.

Increased security: Encapsulation can help to increase the security of your code by hiding the data and co de within a class. This can make it more difficult for attackers to exploit security vulnerabilities in your cod e.

Q7.Is java a t 100% Object oriented Programming language? If no why ?

Ans

Java is not a 100% Object oriented Programming language. This is because it supports all of the core con cepts of OOP, such as encapsulation, inheritance, polymorphism, and abstraction. However, Java also su pports primitive data types, which are not objects. This is why Java is not considered to be a 100% object-oriented programming language.

Here are some of the reasons why Java is not considered to be a 100% object-oriented programming lan guage:

Primitive data types: Java supports primitive data types, such as int, float, and double. These data types a re not objects, and they cannot inherit from other classes or be used in polymorphism.

Static methods and variables: Java allows you to define static methods and variables. Static methods and variables are not associated with any particular object, and they cannot access the data or methods of other objects.

Wrapper classes: Java provides wrapper classes for primitive data types. Wrapper classes are objects that wrap around primitive data types. They allow you to treat primitive data types as objects, but they do not fully support all of the features of OOP.

Overall, Java is a highly object-oriented programming language. However, it does not support all of the features of OOP, and it does support some features that are not considered to be object-oriented. This is why Java is not considered to be a 100% object-oriented programming language.

Q8.What are the advantages of abstraction in java?

Ans

There are several advantages to abstraction in Java:

Reduces complexity: Abstraction can help to reduce the complexity of code by hiding the implementation details of objects. This can make code easier to read and understand, and it can also make it easier to maintain.

Increases reusability: Abstraction can help to increase the reusability of code by creating reusable classes. This can make it easier to build new applications and to add new features to existing applications.

Improves testability: Abstraction can help to improve the testability of code by isolating the implementation details of objects. This can make it easier to write unit tests for code, and it can also make it easier to debug code.

Enhances scalability: Abstraction can help to enhance the scalability of code by allowing you to create different implementations of an object’s functionality. This can make it easier to adapt code to different environments and to different requirements.

Q9.What is an abstraction explained with an Example?

Ans

Abstraction is a concept in object-oriented programming (OOP) that allows you to hide the implementation details of an object from the user. This allows the user to focus on the object’s functionality without having to worry about how it works.

Ex-

```
abstract class Animal {
   private String name;

   public Animal(String name) { this.name = name; }

   public abstract void makeSound();

   public String getName() { return name; }
}
```

```
class Dog extends Animal {
    public Dog(String name) { super(name); }

    public void makeSound()
    {
        System.out.println(getName() + " barks");
    }
}
class Cat extends Animal {
    public Cat(String name) { super(name); }

    public void makeSound()
    {
        System.out.println(getName() + " meows");
    }
}
public class AbstractionExample {
    public static void main(String[] args)
    {
        Animal myDog = new Dog("Buddy");
        Animal myCat = new Cat("Fluffy");

        myDog.makeSound();
        myCat.makeSound();
    }
}
```

Q10.What is the final class in Java?

Ans

The final class is a class that is declared with the final keyword. We can restrict class inheritance by maki
ng use of the final class. Final classes cannot be extended or inherited. If we try to inherit a final class, the
n the compiler throws an error during compilation.