Assignment – 2

1. Program on Single Inheritance

In single inheritance, a class (child class) inherits from another class (parent class).

```
// Parent class
class Animal {
    void eat() {
        System.out.println("This animal eats food.");
    }
}

// Child class inheriting from Animal class
class Dog extends Animal {
    void bark() {
        System.out.println("The dog barks.");
    }
}

public class SingleInheritanceExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.eat();  // Method from parent class
        myDog.bark(); // Method from child class
    }
}
```

2. Program on Multiple Inheritance

Java does not support multiple inheritance with classes to avoid complexity and simplify the design. However, multiple inheritance is supported through interfaces.

```
// First interface
interface Printable {
    void print();
}
```

```
// Second interface
interface Showable {
    void show();
}
// Class implementing both interfaces
class TestMultipleInheritance implements Printable, Showable {
    public void print() {
        System.out.println("Printing...");
    }

    public void show() {
        System.out.println("Showing...");
    }

    public static void main(String[] args) {
        TestMultipleInheritance obj = new TestMultipleInheritance();
        obj.print();
        obj.show();
    }
}
```

3. Why Multiple Inheritance is Not Supported in Java?

Multiple inheritance is not supported in Java with classes because it can lead to ambiguity and complexity. The classic example of the "Diamond Problem" illustrates this issue:

```
class A {
    void display() {
        System.out.println("Display from A");
    }
}

class B extends A {
    void display() {
        System.out.println("Display from B");
    }
}

class C extends A {
    void display() {
        System.out.println("Display from C");
    }
}

// If Java supported multiple inheritance, what would be the output of obj.display()?
// class D extends B, C {
// }

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

In the above scenario, if `D` inherited from both `B` and `C`, calling `obj.display()` would create ambiguity as to which `display()` method to invoke, `B`'s or `C`'s. To avoid this problem, Java supports multiple inheritance only through interfaces.

4. What is `super` and `this` Keyword?

- `this` keyword refers to the current instance of the class.
- `super` keyword refers to the parent class instance.

Example of `this` Keyword

```
class Student {
    int id;
    String name;

    Student(int id, String name) {
        this.id = id;   // 'this' keyword used to refer current instance variables
        this.name = name;
    }

    void display() {
        System.out.println("ID: " + id + ", Name: " + name);
    }
}

public class ThisKeywordExample {
    public static void main(String[] args) {
        Student student = new Student(1, "John");
        student.display();
    }
}
```

Example of `super` Keyword

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}

class Dog extends Animal {
    void sound() {
        super.sound(); // 'super' keyword used to call parent class method
        System.out.println("Dog barks");
    }
}

public class SuperKeywordExample {
    public static void main(String[] args) {
        Dog myDog = new Dog();
        myDog.sound();
    }
}
```

5. Program on Parameterized Constructor

A parameterized constructor is a constructor that takes parameters.

```
class Employee {
    int id;
    String name;

    // Parameterized constructor
    Employee(int id, String name) {
        this.id = id;
        this.name = name;
    }

    void display() {
        System.out.println("Employee ID: " + id + ", Name: " + name);
    }
}

public class ParameterizedConstructorExample {
    public static void main(String[] args) {
        // Creating objects using parameterized constructor
        Employee emp1 = new Employee(101, "Alice");
        Employee emp2 = new Employee(102, "Bob");

        emp1.display();
        emp2.display();
    }
}
```

These examples should help you understand the concepts and implementations of single inheritance, multiple inheritance, the reasons behind Java's design choices regarding inheritance, and the usage of `this` and `super` keywords.