1. **What does the *word* 'polymorphism' mean?**
   Many shapes.

2. **What does it mean when we apply polymorphism to OO design? Give a simple Java example.**
   Providing an object with more than one type through abstract classes or interfaces. It allows the type to behave as if its any of its super class types as well as its own. In the example below, the friend class extends character and implements the IHeal and ITalk interfaces.

```
public class Friend extends Character implements IHeal, ITalk {

    public Friend(String name) {
        super(name);
    }

    public void heal(Player player) {
        int playerCurrentHealth = player.getHealth();
        player.setHealth(playerCurrentHealth + 1);
    }

    @Override
    public String talk() {
        return "I can help you!";
    }
}
```

3. **What can we use to implement polymorphism in Java?**
   This can be achieved through inheritance or interfaces.

4. **How many 'forms' can an object take when using polymorphism?**
   Unlimited

5. **Give an example of when you could use polymorphism.**
   You could have an RPG with a base abstract character class with some basic properties like name, etc. You could have interfaces for different character actions like talk, heal, fight. A friendly character subclass could implement the heal interface, so they can restore the player's health. An enemy subclass could implement the fight interface, so that the player can fight them. An enemy could also have a weakness property that could be used against them in a fight. They can all implement the ITalk interface and a Game class can use this bring back a list of all the characters, friends and enemies.

```java
public class Enemy extends Character implements IFight, ITalk {
    public Weapon weakness;

    public Enemy(String name, Weapon weakness) {
        super(name);
        this.weakness = weakness;
    }

    public Weapon getWeakness() {
        return weakness;
    }

    public String fight(Player player) {
        if (player.getWeapon() == this.weakness){
            return "Player wins";
        }
        return "Player is defeated!";
    }

    @Override
    public String talk() {
        return "I'm going to kill you!";
    }
}
```

```java
public class Game {
    ArrayList<Friend> friends;
    ArrayList<Enemy> enemies;
    Player player;

    public Game(Player player) {
        this.player = player;
        this.friends = new ArrayList<>();
public ArrayList getAllCharacters() {
    ArrayList<ITalk> allCharacters = new ArrayList<>();
    allCharacters.addAll(this.friends);
    allCharacters.addAll(this.enemies);

    return allCharacters;
}
```

6. **What do we mean by 'composition' in reference to object-oriented programming?**
   Composition is a ‘has a’ relationship: one object is composed of other objects.

7. **When would you use composition? Provide a simple example in Java.**
   You could create an RPG Game class and choose how many characters you want to add to it. If you wanted to change the game, you could add more characters to it later.

```
player = new Player( name: "Falconhoof", broadSword,  health: 6);
broadSword = new Weapon( name: "broad sword");
dagger = new Weapon( name: "spoon");
enemy = new Enemy( name: "Charn", broadSword);
friend = new Friend( name: "Eva");
game = new Game(player);
game.addEnemy(enemy);
game.addFriend(friend);
```

8. **What is/are the advantage(s) of using composition?**
   It prevents a higher order class from relying on lower order implementations. It allows you to depend on abstractions, and makes your code more modular. It also prevents messy inheritance chains and allows you to implement only the behaviours that you need.

9. **When an object is destroyed, what happens to all the objects it is composed of?**
   They are destroyed.