Lab1 **Minsi Jämsä**

1 Study following object oriented concepts

1. Object

Object is created to memory and initialized according class definition. Objects created from the same class can have totally different contents, because object can save state etc. information into its member variables. Demonstration code:

```
	Vehicle vehicle = new Vehicle(100);
	vehicle.drive(); //Lab1: 1.1. vehicle Object
```

2. Class

Class defines features (members variables, methods etc.) and implementation which is used in object which is created from the class. Demonstration code:

```
//Lab1: 1.2 Class
public class Vehicle {
	private int speed = 0;

	public Vehicle(int argSpeed) {
		speed = argSpeed;
	}

	public void drive() {
		System.out.println("Vehicle drive() speed is " + speed);
	}
}
```

3. Instantiation of object (creating an object)

When object is created it uses constructor method to initialize its members variables and allocates needed resources like database connections etc. Demonstration code:

```
Vehicle vehicle = new Vehicle(100); // Lab1: 1.3. Instantiation of object (creating an object)
```

4. Visibility (public / private / protected)

Class itself, its methods and member variables can define its visibility. If member variable is public, then it can used and modified by other objects. If member variable is protected then it can used and modified only by classes which inherits the class where member variable is defined. If member variable is private, then only its own class modify its value. Demonstration code:

```
public class Vehicle {
	private int speed = 0; // Lab1: 1.4. Visibility (public / private / protected)

	public void drive() {    // Lab1: 1.4. Visibility (public / private / protected)
		System.out.println("Vehicle drive() speed is " + speed);
	}

	protected int getSpeed() { // Lab1: 1.4. Visibility (public / private / protected)
		return speed;
	}
}
```

5. Member datas / methods

Class can have member variables which are used for saving data for example values . Class methods can be used to modify member variables or doing some other actions like printing values. Demonstration code:

```
public class Vehicle {
            private int speed = 0;     // Lab1: 1.5. Member datas / methods

            public void drive() { // Lab1: 1.5. Member datas / methods
                        System.out.println("Vehicle drive() speed is " + speed);
            }
}
```

6. Inheritance

If multiple classes needs to have something common methods or members, it can be do so that at first base class is created which is then inherited by both classes. For example Vehicle class can be inherited Motorcycle class. Demonstration code:

```
// Lab1: 1.6. Inheritance
public class Motorcycle extends Vehicle {
            public Motorcycle(int argSpeed) {
                        super(argSpeed);
            }
}
```

7. Interface

If there are multiple classes which are different from each others, but they need to be used together, then solution is to define common interface which then all classes implements. Then all different classes can be used together through common interface. Demonstration code:

```
// Lab1: 1.7 Interface
interface PriceInterface {
            public void setPrice(int argPrice);
            public void printPrice();
}

// Lab1: 1.7. Interface
public class Bicycle implements PriceInterface {
        private int bikePrice = 0;

        public void setPrice(int argPrice) {
              bikePrice = argPrice;
        }

        public void printPrice() {
              System.out.println("Bike price is " + bikePrice);
        }
}
```

8. Polymorphism

Polymorphism means that for example class can have methods with same name, but each of them has different arguments. Demonstration code:

```
public class Motorcycle extends Vehicle {
            private double fuelAmount = 0;
```

```
	public void setFuelAmount(int argFuel) { // Lab1: 1.8. Polymorphism
		fuelAmount = (double)argFuel;
	}

	public void setFuelAmount(double argFuel) { // Lab1: 1.8. Polymorphism
		fuelAmount = argFuel;
	}
}
```

9. Overriding

Subclass can override parent class method by creating method which has the same name and arguments as parent class method. Demonstration code:

```
public class Motorcycle extends Vehicle {

	public Motorcycle(int argSpeed) {
		super(argSpeed);
	}

	public void drive() { // Lab1: 1.9. Overriding
		System.out.println("Motorcycle drive() speed is " + super.getSpeed());
	}
}

Motorcycle motorcycle = new Motorcycle(150);
Vehicle motorcycleVehicle = motorcycle;
motorcycleVehicle.drive(); //Lab1: 1.9. Overriding
```

10. Abstract classes

It is not possible to create object from abstract class, because it is basically like a template for the class. Class can then inherit abstract class, but it needs to implement everything which is defined in abstract class, it is similar like interface. Demonstration code:

```
// Lab1: 1.10. Abstract classes
abstract class AbstractBuilding {
	public abstract void printAddress();
}

// Lab1: 1.10. Abstract classes
public class School extends Building {
	public void printAddress() {
		System.out.println("School address is Yliopistokatu 9, 90570 Oulu");
	}
}
```

3. Study Android fundamental concepts

1. What programming languages you can use for Android app development?

Kotlin, Java, and C++ languages.

2. What is .apk file?

When Android application is build, all app contents (files, images etc.) are put into Android Package, APK file. Android devices can then install application from this .apk file.

3. How Android system runs apps?

Android itself is Linux operating system which then runs Android applications. Each application has its own user ID so app can't access other app files etc. Android runs each app in own Linux process which is started when app is used and shutdown when app is closed.

4. Name four types of Android components. Describe each.

- Activities
    - Activity component is starting point of applications user interaction. It is usually device screen with user interface and it can save information what user is/was doing.
- Services
    - Service component is used for running application in background. It does not have user interface.
- Broadcast receivers
    - Broadcast receiver component is used for sending events to application. For example event can tell if device screen is shutdown.
- Content providers
    - Content provider component is used for saving data to device file system. Data can be also for example SQLite database.

5. What is manifest file and what is its purpose?

Manifest file contains information about application. It defines all application components (files). It also defines needed user permissions, Android API level, hardware requirements and libraries.

6. What are resources? Why they are needed?

Resources are needed for avoiding putting everything into application source codes. Application can contain multiple resources. User interfaces are defined in XML files (layout) then user interface can be modified without changing source code (java). Images used in application can be for example png image resources. If application needs to support multiple languages, translations for each language can be defined in XML files.

3.1. UI Hierarchies (Layouts)

Analyze the code you wrote. What it does and why?

Code creates static array of country names which is then used together with ArrayAdapter which creates objects for ListView to show on phone screen.