Certainly! Here are explanations of each concept in OOP in Java with code examples to illustrate each point:

a) Four Basic Features of OOP in Java

1. Encapsulation
    - **Definition**: Encapsulation bundles data and methods within a single unit, a class, and restricts access to them.
    - **Code Example**:

```java
public class Person {
    private String name; // private: restricted access

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

    public String getName() { // public method to access private vari
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

    In this example, `name` is a private variable, only accessible through the public methods `getName` and `setName`.

2. Inheritance
    - **Definition**: Inheritance allows a new class to inherit properties and methods from an existing class.
    - **Code Example**:

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

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

public class Main {
    public static void main(String[] args) {
        Dog dog = new Dog();
        dog.eat(); // Inherited method
        dog.bark(); // Dog's own method
    }
}
```

Here, `Dog` inherits from `Animal`, so it can access `eat()` while also having its own `bark()` method.

3. **Polymorphism**
    - **Definition**: Polymorphism allows objects to respond to the same method call in different ways.
    - **Code Example**:

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

class Cat extends Animal {
    void sound() {
        System.out.println("Meow");
    }
}

class Dog extends Animal {
    void sound() {
        System.out.println("Bark");
    }
}

public class Main {
    public static void main(String[] args) {
        Animal myAnimal = new Dog();
        myAnimal.sound(); // Outputs "Bark" due to polymorphism
    }
}
```

Here, `sound()` behaves differently based on the object type (Cat or Dog).

4. **Abstraction**
    - **Definition**: Abstraction hides complex implementation and exposes only necessary details.
    - **Code Example**:

```java
abstract class Shape {
    abstract void draw(); // Abstract method (no implementation)
}

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

public class Main {
    public static void main(String[] args) {
        Shape shape = new Circle();
        shape.draw(); // Outputs "Drawing a circle"
    }
}
```

Here, `Shape` is abstract and only defines a method `draw` without implementation, while `Circle` provides its specific implementation.

b) Advantages of OOP

1. **Modularity**
   - **Example**: Dividing code into classes, each responsible for its tasks.

```java
class Engine {
    void start() {
        System.out.println("Engine started.");
    }
}

class Car {
    Engine engine = new Engine();

    void drive() {
        engine.start();
        System.out.println("Car is moving.");
    }
}
```

2. Reusability
    - **Example**: Code can be reused through inheritance and polymorphism.

```java
class Vehicle {
    void start() {
        System.out.println("Vehicle started.");
    }
}

class Truck extends Vehicle {
    // Reuses start() from Vehicle
}
```

3. Scalability
    - **Example**: New features or classes can be added easily.

```java
class Animal {
    void eat() {
        System.out.println("Animal eats.");
    }
}

class Bird extends Animal {
    void fly() {
        System.out.println("Bird flies.");
    }
}
```

4. Maintainability
    - **Example**: Encapsulation keeps data safe and manageable.

```java
public class BankAccount {
    private double balance;

    public void deposit(double amount) {
        if (amount > 0) {
            balance += amount;
        }
    }
}
```

5. Improved Productivity
    - **Example**: Using Java’s standard libraries speeds up development.

```
import java.util.ArrayList;

public class Example {
    public static void main(String[] args) {
        ArrayList<String> list = new ArrayList<>(); // Java's pre-bui
        list.add("Hello");
    }
}
```

c) Differences between Class and Object

1. Definition
    - **Class**: Blueprint or template for creating objects.

```
class Car {
    String color;
    void drive() {
        System.out.println("Car is driving.");
    }
}
```

    - **Object**: Instance of a class.

```
Car myCar = new Car(); // `myCar` is an object of class Car
myCar.color = "Red";
```

2. Existence
    - **Class**: Exists in code, does not occupy memory.
    - **Object**: Occupies memory when created.

```
Car myCar = new Car(); // `myCar` takes up memory
```

d) Access Modifiers

1. Public
    - **Definition**: Accessible from any other class.

- **Code Example**:

```java
public class Example {
    public String name = "Public";

    public void display() {
        System.out.println(name);
    }
}
```

2. **Private**
    - **Definition**: Accessible only within the declared class.
    - **Code Example**:

```java
class Example {
    private int age = 25;

    private void showAge() {
        System.out.println(age);
    }
}
```

3. **Protected**
    - **Definition**: Accessible within the same package and by subclasses.
    - **Code Example**:

```java
class Parent {
    protected String familyName = "Smith";
}

class Child extends Parent {
    void displayFamilyName() {
        System.out.println(familyName);
    }
}
```