ENGINEER SERIES

IN JAVA LANGUAGE

4

CH

```
setsize <= NGROUPS_SMALL)
p_info->blocks[0] = group_info->small_block;

for (i = 0; i < nblocks; i++) {
  gid_t *b;
  b = (void *)__get_free_page(GFP_USER);
  if (!b)
    goto out_undo_partial_alloc;
  group_info->blocks[i] = b;
 }
}
```

ABSTRACT CLASS

ENG : Qusay Khudair

Creativity and Accuracy in Work

AUG PLUS .Edu

Chapter 12

Abstract Class

ENG : Qusay khudair

Review of Inheritance

- **Definition:**
  - Inheritance is a mechanism in Java where one class Inherit the properties and behaviors (fields and methods) of another class.
- **Purpose:**
  - To promote code reusability.

Basic Syntax of Inheritance

- **Keywords:**
  - `extends`: Used to indicate that a class is inheriting from a superclass.
  - `Dog extends Animal`: The `Dog` class inherits properties and methods from the `Animal` class.
  - Method Overriding : occurs when a subclass provides a specific implementation for a method that is already defined in its superclass

- **The `super` Keyword : keyword is used to call the superclass's methods or constructors from within a subclass.**

Types of Inheritance in Java

- **Single Inheritance:**
  - **A class inherits from one superclass.**
  - **Example:** `class Dog extends Animal`
- **Multilevel Inheritance:**
  - **A class inherits from a subclass that is also a subclass of another class.**
  - **Example:** `class Cat extends Dog extends Animal`
- **Java <u>does not support</u> multiple inheritance through classes (i.e., a class cannot inherit from more than one class), but it can be achieved using interfaces.**

Abstract Class

- **Definition:**
    - **An Abstract Class is a class that cannot be instantiated on its own. It is designed to be a superclass and can contain abstract methods that must be implemented by subclasses.**
- **Purpose:**
    - **To provide a common base class for other classes to inherit from.**
    - **To define methods that must be implemented by any subclass.**

Abstract Class Syntax

```
abstract class Animal {
    // Abstract method (does not have a body)
    abstract void sound();

    // Regular method
    void sleep() {
        System.out.println("Sleeping...");}}
```

- **Keywords:**
  - **`abstract`: Used to declare a class as abstract or to declare a method without an implementation .**
  - **Note : We can put the keyword abstract before or after access modifier .**
  - **`void sound()`: This method is abstract and must be implemented by any subclass of `Animal`.**

Abstract Methods

- **Definition:**
  - **An Abstract Method is a method that is declared without an implementation (no body).**
- **Purpose:**
  - **To force subclasses to provide specific implementations for certain behaviors.**
  - **Note : No static and private in Abstract Method .**
- **Example:**
  - **In the previous page , the `sound()` method is abstract and must be implemented by any subclass of `Animal`.**

Creating Subclasses

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

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

- **Explanation:**
    - **`Dog` and `Cat` are subclasses of `Animal`.**
    - **Both classes must implement the `sound()` method.**

Using Abstract Classes

```
abstract class Animal {
    // Abstract method (does not have a body)
    abstract void sound();

    // Regular method
    void sleep() {
        System.out.println("Sleeping...");}}
class Dog extends Animal {
    void sound() {
        System.out.println("Bark");}}

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

public class Main {
    public static void main(String[] args) {
        Animal myDog = new Dog();
        Animal myCat = new Cat();
        myDog.sound(); // Outputs: Bark
        myCat.sound(); // Outputs: Meow
        myDog.sleep(); // Outputs: Sleeping...
    }}
```

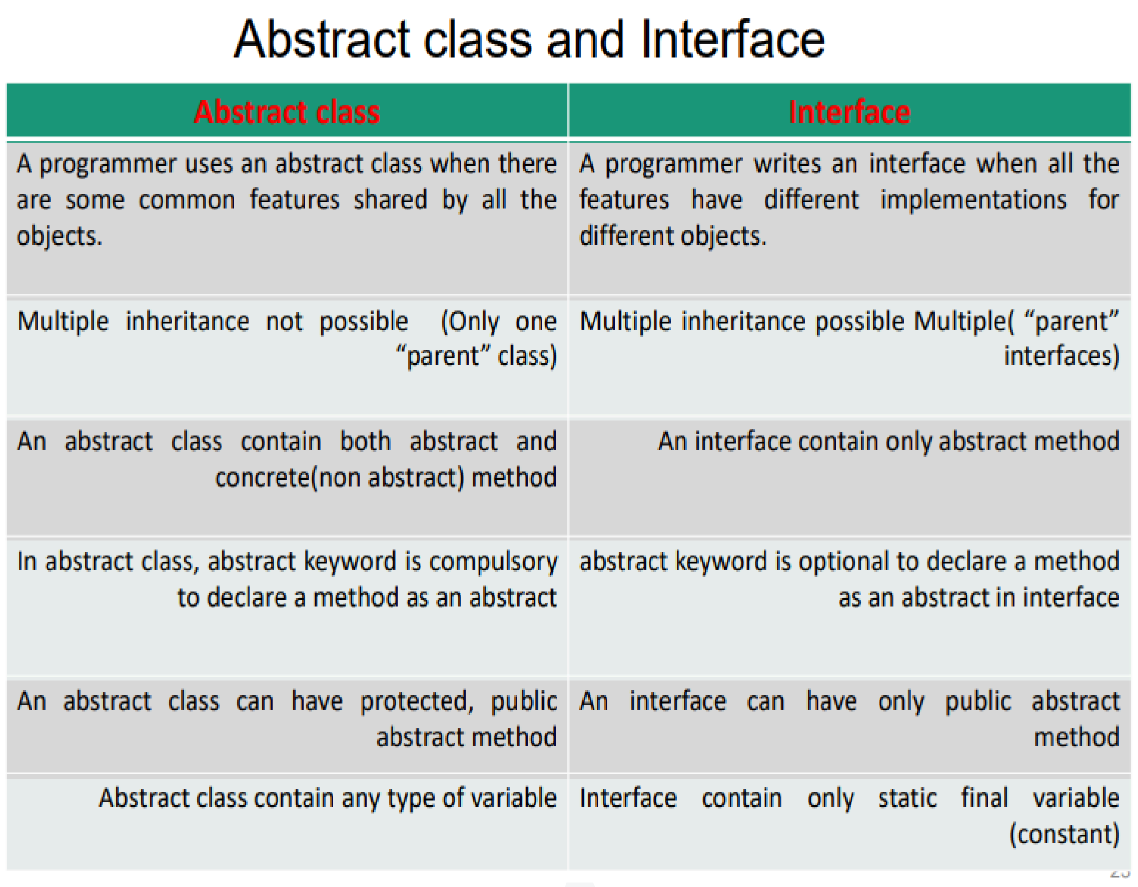
- **Key Points:**
  - **You cannot instantiate an abstract class directly (e.g., `new Animal()` would cause an error).**
  - **Subclasses provide the concrete implementation of abstract methods.**

When to Use Abstract Classes ??

- **Use Abstract Classes:**
  - **When you want to share common code among related classes.**
  - **When you have a base class that should not be instantiated on its own.**
  - **When you need some methods to be implemented differently by different subclasses.**
- **Examples:**
  - **Abstract classes are commonly used in frameworks and libraries where a common base class is provided with some default behavior and abstract methods for specific behaviors.**

Abstract Class vs. Interface

- **Abstract Class:**
  - **Can have both abstract and concrete methods.**
  - **Can have instance variables and constructors.**
  - **Inherited using `extends`.**
- **Interface:**
  - **All methods are abstract by default (before Java 8).**
  - **Cannot have instance variables.**
  - **Implemented using `implements`.**



Abstract class and Interface

Abstract class	Interface
A programmer uses an abstract class when there are some common features shared by all the objects.	A programmer writes an interface when all the features have different implementations for different objects.
Multiple inheritance not possible (Only one "parent" class)	Multiple inheritance possible Multiple("parent" interfaces)
An abstract class contain both abstract and concrete(non abstract) method	An interface contain only abstract method
In abstract class, abstract keyword is compulsory to declare a method as an abstract	abstract keyword is optional to declare a method as an abstract in interface
An abstract class can have protected, public abstract method	An interface can have only public abstract method
Abstract class contain any type of variable	Interface contain only static final variable (constant)

Full Example :

```
abstract class Employee {
 private String name;
 private double salary;
public Employee(String name, double salary) {
this.name = name;
 this.salary = salary; }

 public String getName() {
return name; }

public double getSalary() {
return salary; }
 // Abstract method to be implemented by subclasses
 public abstract double calculateBonus();
 public void displayDetails() {
System.out.println("Employee Name: " + name);
System.out.println("Employee Salary: " + salary);
 } }

class Manager extends Employee {

    public Manager(String name, double salary) {
        super(name, salary);
    }

    @Override
    public double calculateBonus() {
        return getSalary() * 0.20;
    }
}
```

```
class Developer extends Employee {

    public Developer(String name, double salary) {
        super(name, salary); }

    @Override
    public double calculateBonus() {
        return getSalary() * 0.10;
    }
}

public class Company {
    public static void main(String[] args) {
        // Create Manager object
        Employee manager = new Manager("Alice",
90000);
        manager.displayDetails();
        System.out.println("Manager Bonus: " +
manager.calculateBonus());

        System.out.println();

        // Create Developer object
        Employee developer = new Developer("Bob",
80000);
        developer.displayDetails();
        System.out.println("Developer Bonus: " +
developer.calculateBonus());}}
```

```
Output
Employee Name: Alice
Employee Salary: 90000.0
Manager Bonus: 18000.0

Employee Name: Bob
Employee Salary: 80000.0
Developer Bonus: 8000.0
```
