OOP Concepts: Polymorphism, Method Overloading, Encapsulation

# 1. Difference between Method Overloading and Method Overriding

• Method Overloading:  
 - Occurs within the same class (can appear across inheritance if signatures differ).  
 - Same method name, but different parameter lists (number or type of arguments).  
 - Decided at compile-time (compile-time polymorphism).  
 - Inheritance not required.  
  
Example:  
class MathUtils {  
 int add(int a, int b) { return a + b; }  
 double add(double a, double b) { return a + b; } // overloaded  
}  
  
• Method Overriding:  
 - Happens across inheritance (parent → child).  
 - Child provides a new implementation for a parent method.  
 - Same method name, parameters, and return type (or covariant return).  
 - Resolved at runtime (runtime polymorphism).  
 - Inheritance is mandatory.  
  
Example:  
class Animal {  
 void sound() { System.out.println("Animal makes sound"); }  
}  
class Dog extends Animal {  
 @Override  
 void sound() { System.out.println("Dog barks"); }  
}

# 2. Can you inherit a private method? If not, why?

No. Private methods are not inherited by child classes.  
Reason: private members are only accessible within the class they are defined in.  
The child class has no visibility of them, so they cannot be overridden.  
  
Example:  
class Parent {  
 private void secretMethod() {  
 System.out.println("This is private");  
 }  
}  
class Child extends Parent {  
 // Cannot access secretMethod() here  
}

# 3. Can a final class be inherited?

No. A final class cannot be extended.  
Purpose: Prevents modification or extension (e.g., java.lang.String is final).  
  
Example:  
final class Vehicle {  
 void run() { System.out.println("Vehicle is running"); }  
}  
// This will cause an error:  
class Car extends Vehicle { } // ❌ cannot inherit from final class

# 4. What happens if a parent class reference points to a child class object?

This is runtime polymorphism (dynamic method dispatch).  
• Parent reference can hold a child object.  
• If a method is overridden, the child’s version runs.  
• If not overridden, parent’s version runs.  
• Parent reference cannot directly access child-specific methods without casting.  
  
Example:  
class Animal {  
 void sound() { System.out.println("Animal sound"); }  
}  
class Dog extends Animal {  
 @Override  
 void sound() { System.out.println("Dog barks"); }  
 void wagTail() { System.out.println("Dog wags tail"); }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Animal a = new Dog();  
 a.sound(); // Dog barks  
 // a.wagTail(); // ❌ compile error  
 }  
}

# 5. Can you access the parent class constructor from a child class?

Yes, using the super() keyword.  
• super() is used inside the child class constructor to explicitly call the parent’s constructor.  
• If not called, Java automatically calls the default parent constructor.  
• If the parent does not have a no-arg constructor, super(args) must be used.  
  
Example:  
class Parent {  
 Parent(String name) {  
 System.out.println("Parent constructor: " + name);  
 }  
}  
  
class Child extends Parent {  
 Child(String name, int age) {  
 super(name);  
 System.out.println("Child constructor: age = " + age);  
 }  
}  
  
public class Main {  
 public static void main(String[] args) {  
 Child c = new Child("John", 25);  
 }  
}