**Abstract dan Polymorphism**

**Tugas 4 PEMPROGRAMAN LANJUT**

Nama Anggota Kelompok :

Abmi Sukma Edri : 12250120341

Daffa Ikhwan Nurfauzan : 12250110979

Dwi Jelita Adhliyah : 12250120331

1. **Abstrak**
2. Aturan Class Abstract

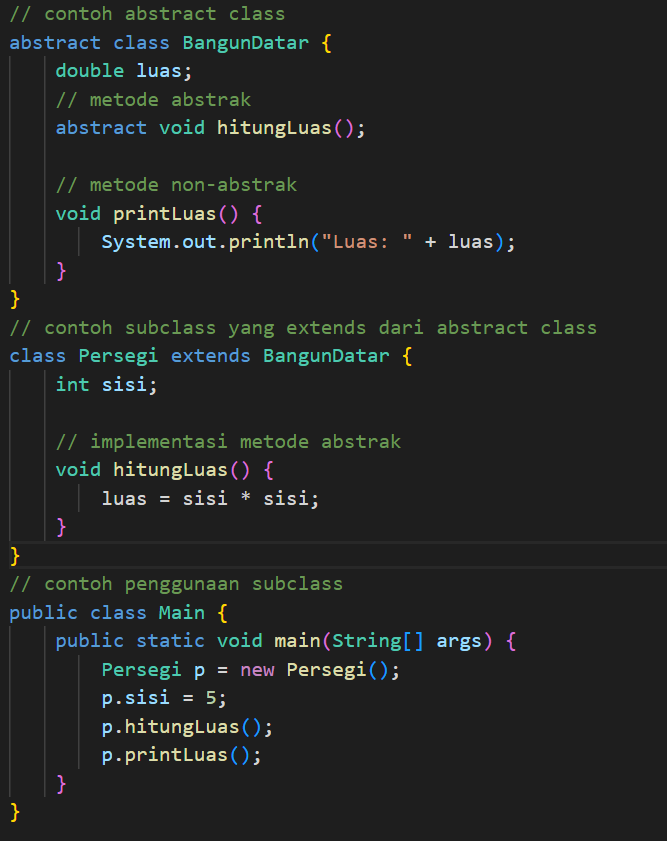
Class abstrak adalah class yang masih dalam bentuk abstrak. Karena bentuknya masih abstrak, dia tidak bisa dibuat langsung menjadi objek.

Sebuah class agar dapat disebut class abstrak setidaknya memiliki satu atau lebih method abstrak. Method abstrak adalah method yang tidak memiliki implementasi atau tidak ada bentuk konkritnya atau method yang tidak memiliki isi, hanya terdapat nama saja.

Aturan class abstrak:

1. Tidak dapat diinstansiasi langsung.
2. Memiliki minimal satu metode abstrak.
3. Dapat memiliki metode non-abstrak.

Contoh penggunaan class abstrak dalam bahasa pemrograman Java:

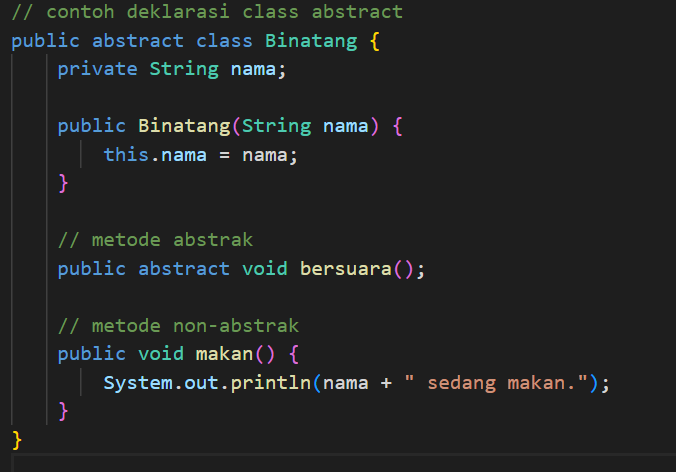


Pada contoh di atas, BangunDatar adalah class abstrak yang memiliki metode abstrak hitungLuas(), serta metode non-abstrak printLuas(). Subclass Persegi yang extends dari BangunDatar harus mengimplementasikan metode abstrak hitungLuas() dan dapat menggunakan metode non-abstrak printLuas() yang diwarisi dari BangunDatar. Dalam main method, kita membuat objek Persegi dan mengisi nilai sisi, kemudian memanggil hitungLuas() untuk menghitung luas persegi dan printLuas() untuk mencetak nilai luas.

1. Mendeklarasikan class Abstrak

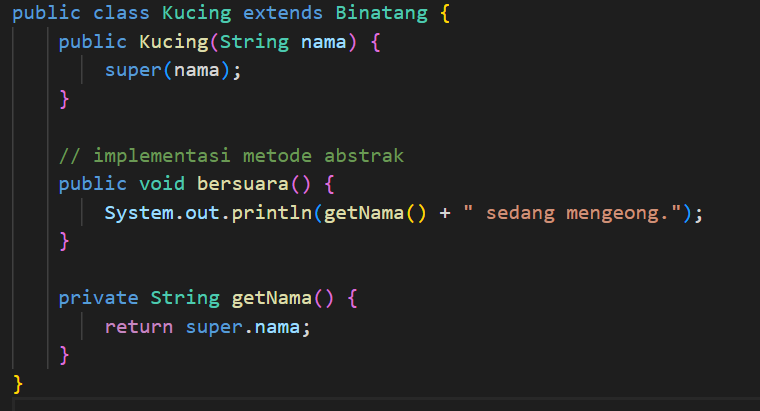
Mendeklarasikan sebuah class abstract dengan menambahkan kata kunci abstract di depan kata kunci class pada saat mendefinisikan sebuah class. Pada dasarnya, class abstract merupakan sebuah class yang tidak dapat diinstansiasi, tetapi digunakan sebagai kerangka untuk subclass yang akan mengimplementasikan atau mewarisi perilaku dari class abstract tersebut.

Berikut ini adalah contoh deklarasi class abstract dalam bahasa pemrograman Java:



Pada contoh di atas, class Binatang merupakan sebuah class abstract yang memiliki constructor untuk menginisialisasi nama binatang, serta memiliki metode abstrak bersuara() yang harus diimplementasikan oleh subclass dan metode non-abstrak makan() yang dapat digunakan oleh subclass.

Sebagai contoh, kita dapat membuat subclass Kucing yang extends dari Binatang dan mengimplementasikan metode abstrak bersuara():

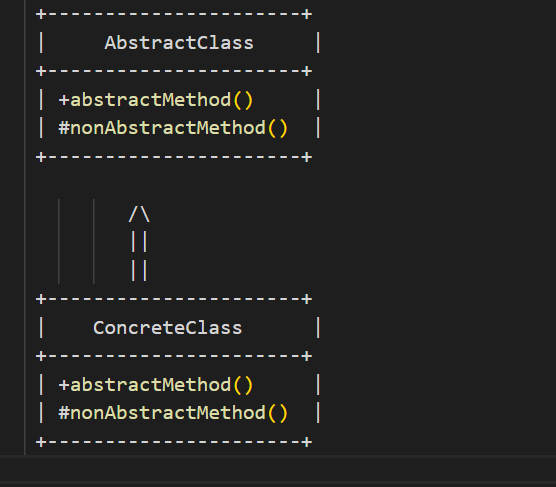


Dalam contoh tersebut, kita membuat subclass Kucing yang extends dari Binatang, kemudian mengimplementasikan metode abstrak bersuara() yang telah didefinisikan di class Binatang. Kita juga dapat menggunakan metode non-abstrak makan() yang diwarisi dari Binatang.

Perlu diingat bahwa jika sebuah class memiliki setidaknya satu metode abstrak, maka class tersebut harus dideklarasikan sebagai class abstract. Jika terdapat sebuah class yang extends dari sebuah class abstract, maka subclass tersebut harus mengimplementasikan seluruh metode abstrak dari superclass-nya atau dideklarasikan kembali sebagai class abstract.

1. Class Diagram Abstract

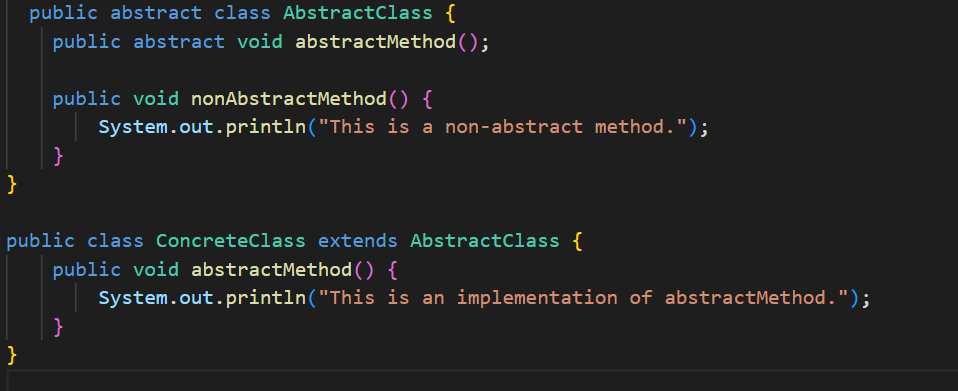
**Class diagram**adalah**jenis diagram struktur statis dalam UML yang menggambarkan struktur sistem dengan menunjukkan sistem class, atributnya, metode, dan hubungan antar objek.** Class diagram disebut jenis diagram struktur karena menggambarkan apa yang harus ada dalam sistem yang dimodelkan dengan berbagai komponen



Pada diagram di atas, terdapat sebuah abstract class bernama AbstractClass yang memiliki satu method abstrak (abstractMethod()) dan satu method non-abstrak (nonAbstractMethod()). Class ini tidak dapat di-instantiate secara langsung, tetapi dapat digunakan sebagai kerangka atau blueprint untuk subclass.

Terdapat sebuah subclass dari AbstractClass bernama ConcreteClass yang extends dari AbstractClass. Class ini harus mengimplementasikan method abstrak abstractMethod() dari superclass-nya dan dapat menggunakan method non-abstrak nonAbstractMethod() yang diwarisi dari AbstractClass.

Sebagai contoh, kita dapat membuat implementasi dari abstract class tersebut dalam bahasa pemrograman Java sebagai berikut:



Dalam contoh di atas, AbstractClass adalah sebuah abstract class dengan satu method abstrak abstractMethod() dan satu method non-abstrak nonAbstractMethod(). ConcreteClass adalah sebuah subclass yang extends dari AbstractClass dan mengimplementasikan method abstrak abstractMethod() dari superclass-nya.

1. Keuntungan method abstrak

Keuntungan dari penggunaan method abstrak adalah sebagai berikut:

1. Menentukan interface: Method abstrak dapat digunakan untuk menentukan interface atau kontrak yang harus diikuti oleh subclass. Dengan demikian, subclass harus mengimplementasikan method abstrak tersebut untuk memastikan bahwa subclass mematuhi kontrak yang telah ditentukan.
2. Peningkatan fleksibilitas: Dengan menggunakan method abstrak, kita dapat meningkatkan fleksibilitas dalam perancangan program. Kita dapat menentukan perilaku yang sama di antara berbagai subclass, sementara pada saat yang sama mengizinkan perilaku yang berbeda pada subclass yang berbeda.
3. Kode yang lebih ringkas: Dengan menggunakan method abstrak, kita dapat membuat kode yang lebih ringkas dan efisien. Kita dapat menempatkan kode umum pada method abstrak, sementara pada saat yang sama memungkinkan subclass untuk menambahkan kode tambahan yang sesuai dengan kebutuhan mereka.
4. **POLYMORPHIL**
5. Apa itu Polymorphism?

Polimorfisme dalam OOP adalah sebuah prinsip di mana class dapat memiliki banyak **“bentuk”** method yang berbeda-beda meskipun namanya sama. “Bentuk” di sini dapat kita artikan: isinya berbeda, parameternya berbeda, dan tipe datanya berbeda.

1. Bagaimana Membuat Polymorphism?

Polimorfisme dapat dicapai dalam pemrograman objek dengan dua cara: melalui penggunaan overloading dan overriding.

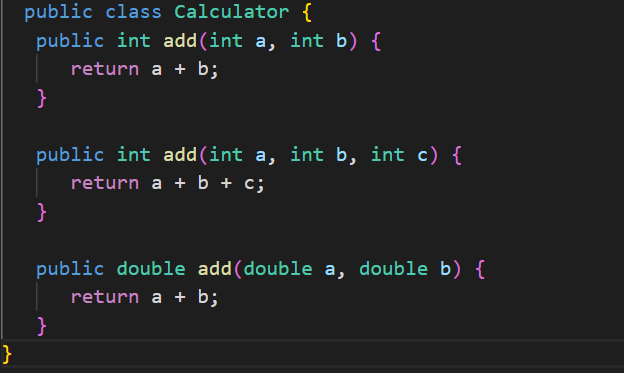
Polimorfisme pada Java ada dua macam:

1. Static Polymorphism (Polimorfisme statis);
2. Dynamic Polymorphism (Polimorfisme dinamis).

Beda dari keduanya terletak pada cara membuat polimorfismenya. Polimorfisme statis menggunakan method overloading sedangkan polimorfisme dinamis menggunakan method overriding.

1. Overloading

Method overloading terjadi pada sebuah class yang memiliki nama method yang sama tapi memiliki parameter dan tipe data yang berbeda. Dalam hal ini, compiler akan memilih method yang sesuai berdasarkan jumlah, urutan, atau tipe parameter yang digunakan dalam pemanggilan method.

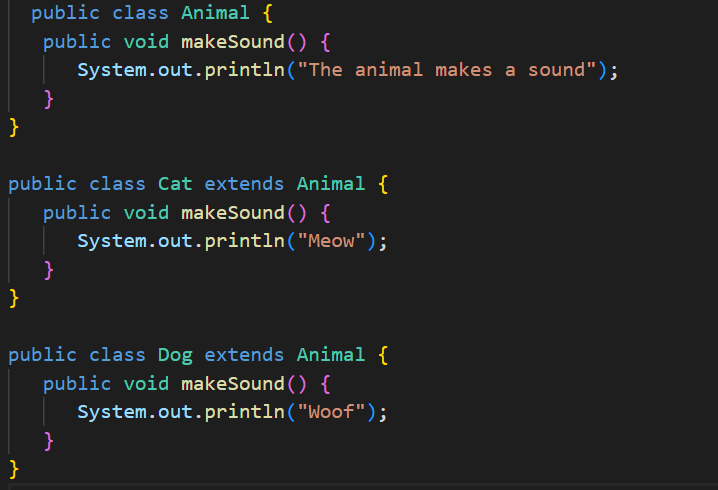


Dalam contoh di atas, kelas Calculator memiliki tiga method add() yang berbeda, yang dapat dipanggil dengan berbagai cara tergantung pada jumlah dan tipe parameter yang digunakan.

1. Overriding

Overriding terjadi ketika sebuah subclass memiliki metode dengan nama, parameter, dan tipe pengembalian yang sama dengan metode pada superclass-nya. Dalam hal ini, metode pada subclass akan menimpa atau menggantikan metode pada superclass-nya.

Contoh:



Dalam contoh di atas, kelas Animal adalah superclass, sedangkan Cat dan Dog adalah subclass. Subclass Cat dan Dog memiliki metode makeSound() yang sama dengan superclass-nya, namun memiliki implementasi yang berbeda. Ketika metode makeSound() dipanggil pada objek Cat atau Dog, metode yang dipanggil akan tergantung pada tipe objek yang sebenarnya sedang diproses pada saat runtime.

Dengan demikian, polimorfisme dapat dicapai dengan mengandalkan fitur overloading dan overriding pada pemrograman objek.

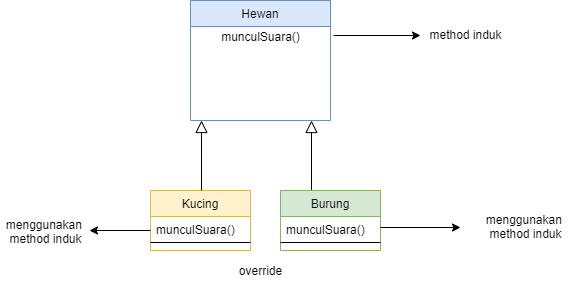
1. Polymer Dinamis

Polimorfisme dinamis (dynamic polymorphism) atau disebut juga late binding adalah bentuk polimorfisme yang diselesaikan pada waktu runtime. Disini programmer mendefinisikan berbagai method dengan nama dan jumlah parameter yang sama. Ketika method dipanggil, JVM yang akan memutukan method mana yang dipanggil melalui aturan-aturan yang telah ditetapkan oleh Java.

Polimorfisme dinamis identik dengan menggunakan (inheritance), implementasi interface bahkan abstrak class. Pada polimorfisme dinamis ini menggunakan *method overriding*.

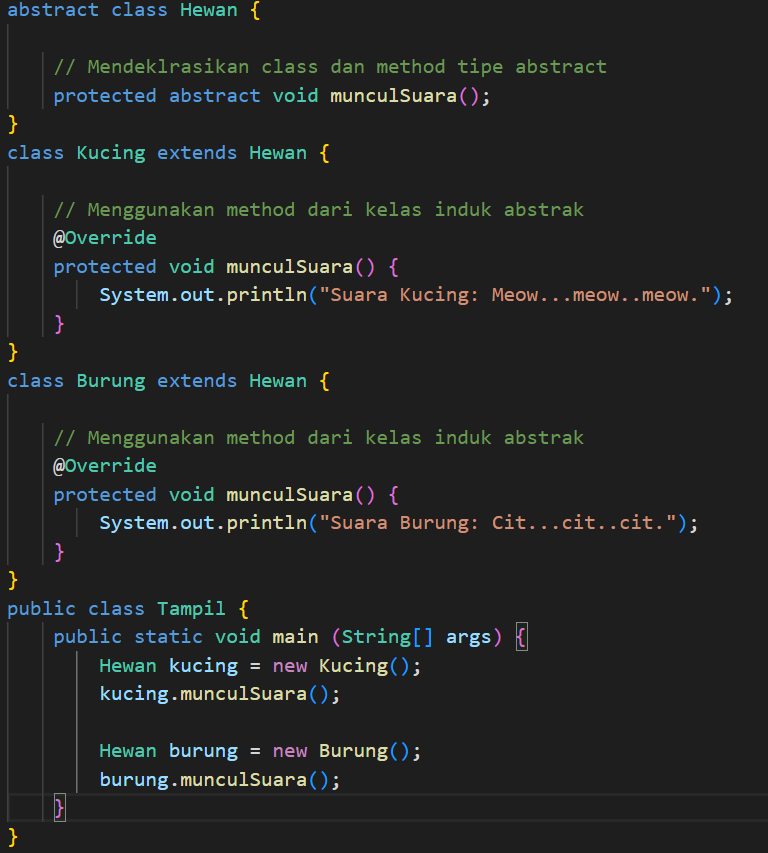
Contoh :

Perhatikan gambar tabel *class* diagram di bawah ini.



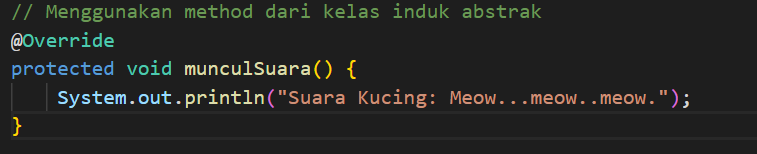
Kita akan coba membuat program sesuai dengan gambar tabel di atas.

Buatlah *class***Hewan.java**. Berarti ini merupakan kelas induk dari semua kelas. Kemudian buat *class* **Kucing.java** dan **Burung.java** sebagai anaknya. Kamu bisa membuatnya dalam satu *class* maupun *class*yang berbeda.



**Class Hewan.java memiliki method utama yaitu munculSuara(). Kita menambahkan beberapa subclass anak kelas dari Hewan.java yakni Kucing.java dan Burung.java. Mereka memiliki method yang sama meskipun menampilkan statements argumen yang berbeda.**

**Kucing.java**

 **Burung.java**

### 

### Aturan Method Overriding

* Mode akses overriding method harus sama atau lebih luas daripada override method.
* Subclass hanya dapat dan boleh meng-override method superclass satu kali saja. Tidak boleh ada lebih dari satu method yang sama pada kelas.
* Soal aturan hak akses, setiap *subclass* tidak boleh mempunyai hak akses method overriding yang ketat dibandingkan dengan hak akses method pada superclass ataupun parent class.

1. Polymer stactic

Polimorfisme statis (static polymorphism) atau disebut juga early binding adalah bentuk polimorfisme yang diselesaikan pada saat compile. Disini seorang programmer mendefinisikan berbagai method dengan nama yang sama namun memiliki jumlah parameter yang berbeda-beda. Ketika method dipanggil, programmer dapat mengetahui dengan jelas method mana yang akan dijalankan sesuai dengan jumlah parameternya.

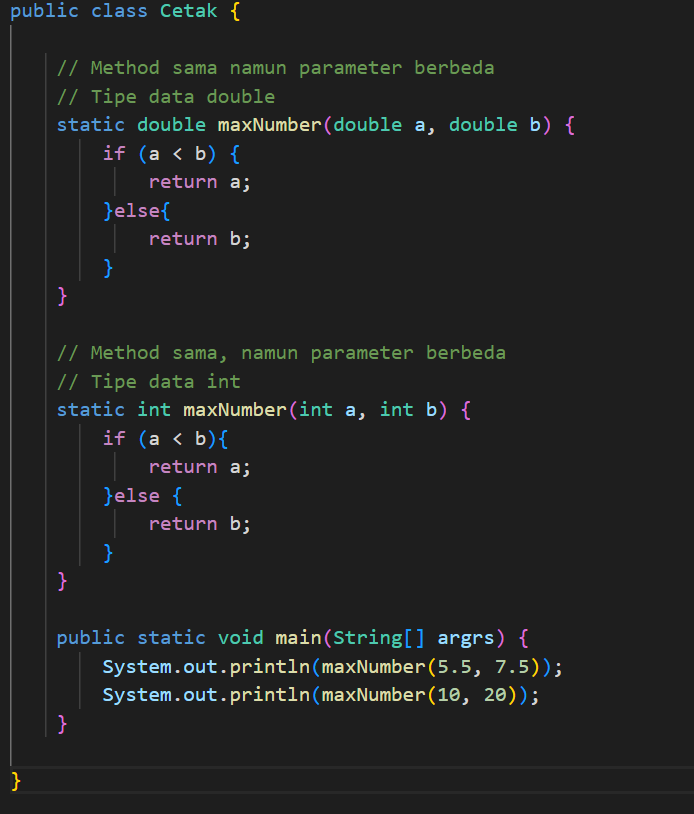
polimorfisme statis hanya terjadi dalam satu class saja. Pada polimorfisme static ini menggunakan *method overloading.*

**Aturan Method Overloading**

* Nama method harus sama dengan method lainnya.
* Parameter haruslah berbeda.
* Return boleh sama, juga boleh berbeda.

Contoh:

Misal kamu membuat sebuah *class*dengan nama **Cetak.java**. Pada *class* ini mempunyai method maxNumber(). Perhatikan kode program dibawah ini.



Pada *class***Cetak.java**memiliki 2 *method* yang sama yaitu **maxNumber()**. Tapi parameter dan tipenya berbeda, yaitu:

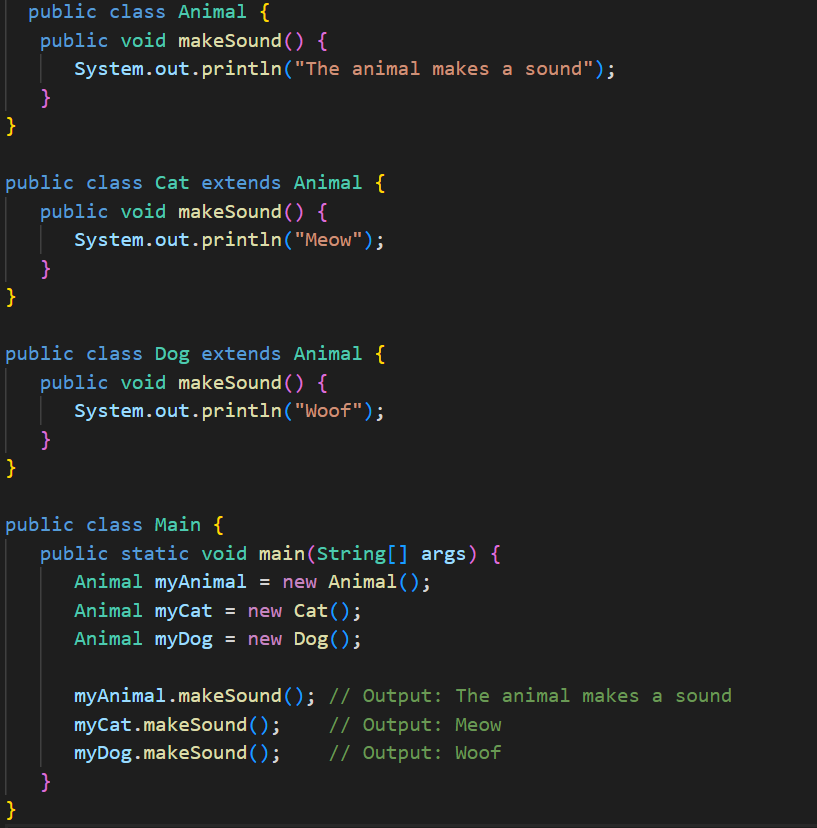
* static double maxNumber(double a, double b)
* static int maxNumber(int a, int b)

Yang pertama memiliki parameter dan tipe data *double,*sedangkan satunya lagi memiliki parameter dan tipe data *int.*Hal ini jelas berbeda

1. Polymorphism dalam Pewarisan?

Polimorfisme dalam pewarisan dapat dicapai dengan menggunakan overriding pada subclass. Ketika sebuah metode yang di-overriding dipanggil pada objek yang merupakan instance dari subclass, maka metode yang dipanggil adalah metode pada subclass, bukan pada superclass.

Contoh:

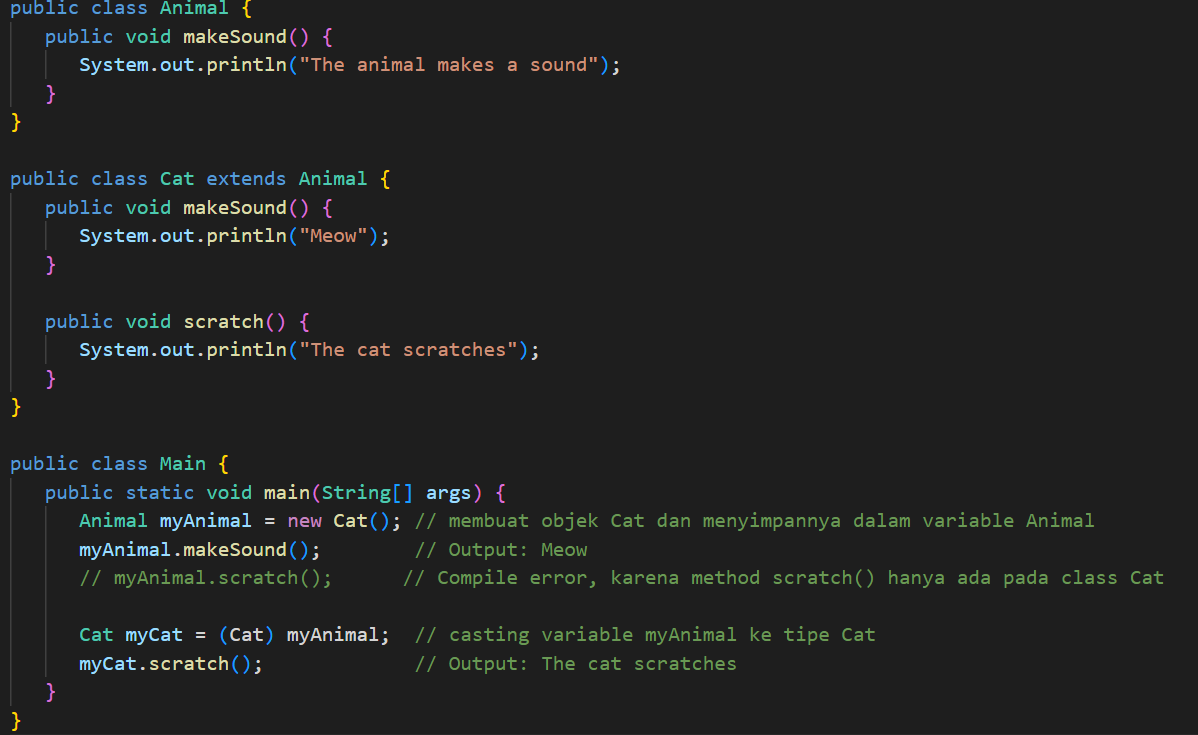


Dog merupakan subclass dari kelas Animal. Ketika metode makeSound() di-overriding pada kelas Cat dan Dog, metode tersebut memiliki implementasi yang berbeda dari metode pada superclass-nya. Pada saat runtime, ketika metode makeSound() dipanggil pada objek myCat dan myDog, yang dipanggil adalah metode makeSound() pada subclass masing-masing, karena objek-objek tersebut merupakan instance dari subclass. Hal ini menunjukkan polimorfisme dalam pewarisan, di mana objek-objek dapat di-deklarasikan sebagai tipe superclass, namun metode yang dipanggil pada objek-objek tersebut dapat bervariasi tergantung pada jenis objek yang sebenarnya sedang diproses pada saat runtime.

1. Variable superclass dapat memegang referensi dari objek subclass

Variable superclass dapat memegang referensi dari objek subclass melalui polimorfisme. Ketika sebuah objek subclass di-deklarasikan sebagai tipe superclass, objek tersebut dapat diakses melalui variable superclass yang merujuk pada objek tersebut. Namun, variable tersebut hanya dapat mengakses metode dan atribut yang didefinisikan pada superclass. Untuk mengakses metode atau atribut yang hanya ada pada subclass, variable harus di-casting ke tipe subclass.

Contoh:



Dalam contoh di atas, sebuah objek Cat di-deklarasikan sebagai tipe Animal dan disimpan dalam variable myAnimal. Ketika metode makeSound() dipanggil pada variable myAnimal, yang dipanggil adalah metode makeSound() pada subclass Cat, karena objek tersebut sebenarnya adalah instance dari Cat. Namun, ketika metode scratch() dipanggil pada variable myAnimal, akan terjadi compile error, karena metode tersebut hanya ada pada subclass Cat.

Untuk mengakses metode scratch(), variable myAnimal harus di-casting ke tipe Cat terlebih dahulu, seperti yang ditunjukkan pada baris 14. Setelah variable di-casting, metode scratch() dapat diakses pada objek yang sebenarnya, yaitu objek Cat.