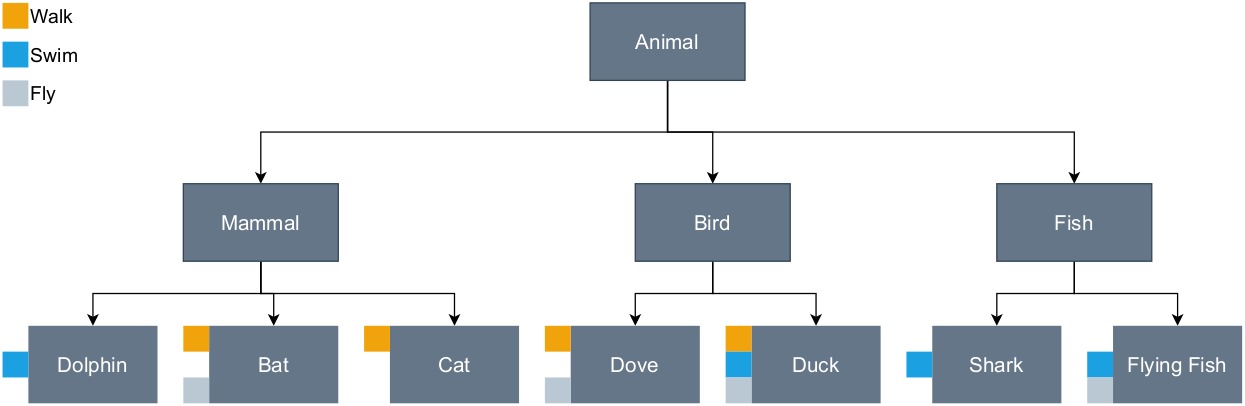
**Mixins**

***Mixin*** adalah cara menggunakan kembali kode kelas dalam banyak hirarki kelas. Konsep *mixin* mungkin adalah konsep yang baru bagi Anda karena konsep ini tidak ada pada bahasa C# atau Java. Jadi kenapa dan kapan kita perlu menggunakan *mixin*?

Kita kembali pada contoh hewan. Perhatikan diagram berikut:

[](https://www.dicoding.com/academies/191/tutorials/7633)

Kita memiliki *superclass* Animal dengan tiga *subclass*. Di bawahnya ada beberapa kelas turunan yang memiliki perilaku berbeda-beda. Beberapa hewan memiliki perilaku yang sama, seperti Cat dan Duck sama-sama bisa berjalan. Kita bisa saja membuat kelas seperti Walkable, Swimmable, dan Flyable. Sayangnya, Dart tidak mendukung *multiple inheritance*, sehingga sebuah kelas hanya bisa mewarisi (*inherit*) satu kelas induk. Kita bisa saja membuat *interface* lalu mengimplementasikannya ke ke kelas Cat atau Duck. Namun, implementasi *interface* mengharuskan kita untuk meng-*override* *method* dan membuat implementasi fungsi di masing-masing kelas.

1. mixin Flyable {
2. void fly() {
3. print("I'm flying");
4. }
5. }
7. mixin Walkable {
8. void walk() {
9. print("I'm walking");
10. }
11. }
13. mixin Swimmable {
14. void swim() {
15. print("I'm swimming");
16. }
17. }

Kelas *mixin* dapat didefinisikan dengan *keyword* class seperti kelas pada umumnya. Jika Anda tidak ingin kelasnya bertindak seperti kelas biasa misalnya seperti bisa diinstansiasi menjadi objek, gunakan saja *keyword* mixin. Setelah membuat kelas seperti di atas kita bisa menambahkan sebagai *mixin* dengan *keyword* with dan diikuti dengan satu atau beberapa kelas *mixin*.

1. class Cat extends Mammal with Walkable { }
3. class Duck extends Bird with Walkable, Flyable, Swimmable { }

Dengan *mixin* ini memungkinkan objek cat untuk memanggil metode walk(). Sementara objek duck bisa memanggil metode walk(), fly(), dan swim().

1. void main() {
2. var donald = Duck();
3. var garfield = Cat();
5. garfield.walk();
7. donald.walk();
8. donald.swim();
9. }

Jika diperhatikan *mixin* ini memang mirip dengan *multiple inheritance*. Namun kelas *mixin* ini tidak termasuk ke dalam hirarki *parent-child* atau *inheritance*. Oleh sebab itu *mixin* memungkinkan kita terhindar dari masalah yang sering terjadi pada *multiple inheritance* yang dikenal dengan [diamond problem](https://en.wikipedia.org/wiki/Multiple_inheritance#The_diamond_problem), yaitu ada dua *parent class* yang memiliki *method* dengan nama yang sama sehingga *child class*-nya ambigu dalam menjalankan *method* yang mana.

Sebagai contoh kita punya kelas bernama Performer.

1. abstract class Performer {
2. void perform();
3. }

Lalu kita punya dua kelas turunan dari Performer.

1. class Dancer extends Performer {
2. @override
3. void perform() {
4. print('Dancing');
5. }
6. }
8. class Singer extends Performer {
9. @override
10. void perform() {
11. print('Singing');
12. }
13. }

Kita asumsikan Dart memiliki dukungan terhadap *multiple inheritance* sehingga kita punya 1 kelas lagi seperti berikut:

1. class Musician extends Dancer, Singer {
2. void showTime() {
3. perform();
4. }
5. }

Kira-kira *method* perform() mana yang akan dijalankan? Beruntung dengan Dart kita bisa menghindari situasi seperti ini dengan mixin.

Ketika mencampur (*mixing*) kelas, kelas yang digunakan sebagai *mixin* tidak paralel namun saling bertumpuk. Itulah mengapa kelas atau method pada *mixin* tidak ambigu satu sama lain. Karena itu jugalah, urutan menjadi hal yang penting dalam menerapkan *mixin*. Misalnya kita telah menerapkan *mixin* pada kelas Musician.

1. mixin Dancer implements Performer {
2. @override
3. void perform() {
4. print('Dancing');
5. }
6. }
8. mixin Singer implements Performer {
9. @override
10. void perform() {
11. print('Singing');
12. }
13. }
15. class Musician extends Performer with Dancer, Singer {
16. void showTime() {
17. perform();
18. }
19. }

Lalu buatlah objek yang akan menjalankan *method* perform().

1. void main() {
2. var arielNoah = Musician();
3. arielNoah.perform();
4. }

Coba jalankan fungsi main di atas, apakah yang akan tampil pada konsol? Mengapa demikian? Seperti yang telah dijelaskan, kelas *mixin* bersifat *stack* atau bertumpuk. Kelas-kelas ini berurutan dari yang paling umum hingga paling spesifik. Sehingga sesuai urutan *mixin* di atas kelas Musician akan menampilkan *method* dari Singer karena berada di urutan terakhir atau paling spesifik.

[](https://www.dicoding.com/academies/191/tutorials/7633)