Polymorphism: dynamic_cast and typeid

```
/*

    . Topics:
        . Dynamic casts and typeid operator:
            .#1: Dynamic casts
            .#2: typeid operator
*/
```

Polymorphism: dynamic_cast

```
Animal
 ├──> virtual ~Animal()                          // Virtual destructor
 ├──> virtual void breathe() const               // Base breathe()

Feline : public Animal
 ├──> virtual ~Feline()                          // Virtual destructor
 ├──> virtual void run() const                   // Base run()
 ├──> void do_some_feline_thingy() const        // Non-virtual method

Dog : public Feline
 ├──> virtual ~Dog()                             // Virtual destructor
 ├──> virtual void bark() const                  // Base bark()
 ├──> void do_some_dog_thingy() const           // Non-virtual method
```

Polymorphism: dynamic_cast

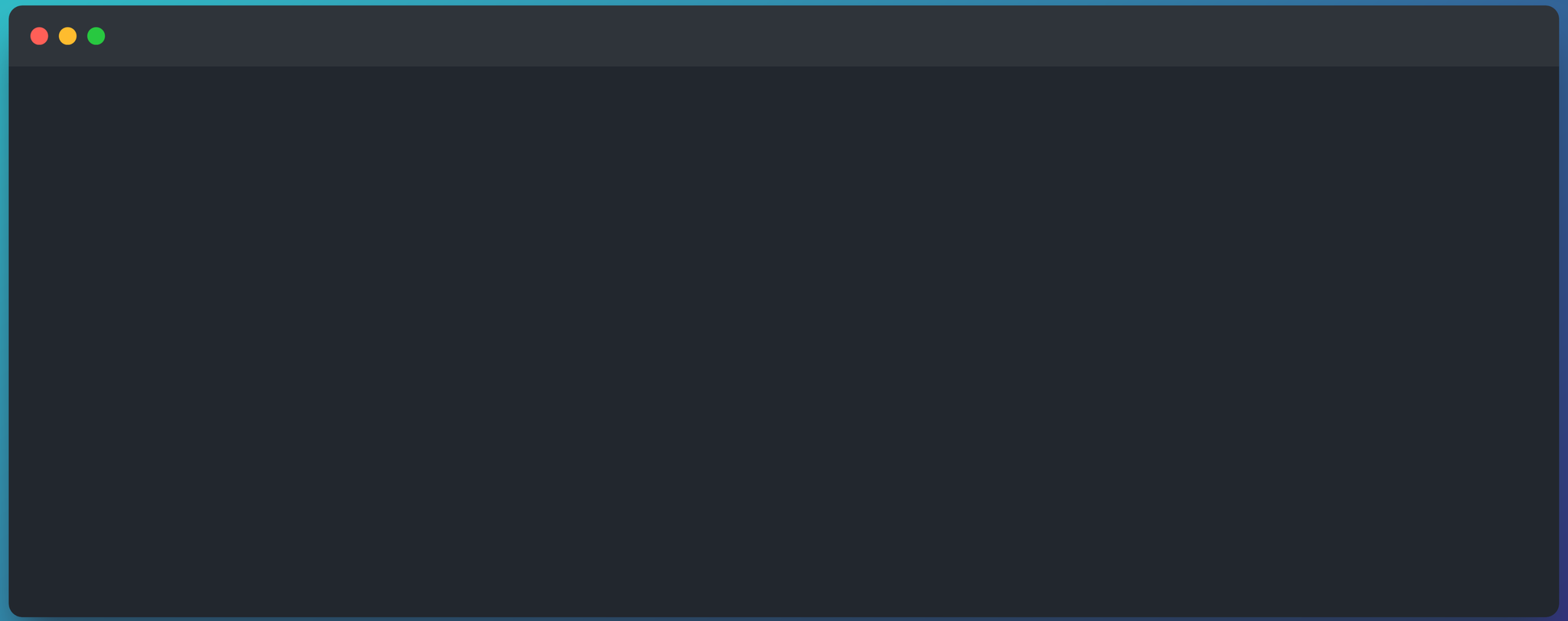
```
// Can't call a derived non-virtual function on a base class pointer
Animal *animal1 = new Feline("stripes", "feline1");
//animal1->do_some_feline_thingy(); // Won't work. do_some_feline_thingy() is not a virtual function coming  from Animal.



//You can do that if you cast to a derived class pointer through dynamic_cast for virtual objects
// If the cast succeeds, we get a valid pointer back,
// if it fails, we get nullptr. So we can check before
// calling stuff on the returned pointer
Feline *feline_ptr = dynamic_cast<Feline *>(animal1);

if (feline_ptr) {
  feline_ptr->do_some_feline_thingy();
} else {
  fmt::println("Couldn't do the transformation from Animal* to Dog*");
}
```

Demo time!



Polymorphism: typeid

```cpp
// 2. typeid with references(polymorphic)
fmt::println("Polymorphic references : ");
inh_poly_2::DynamicDerived dynamic_derived;
inh_poly_2::DynamicBase &base_ref = dynamic_derived;
fmt::println("Type of dynamic_derived : {}", typeid(dynamic_derived).name());
fmt::println("Type of base_ref : {}", typeid(base_ref).name());

fmt::println("----------------");

// 3. typeid with pointers(polymorphic)
fmt::println("Polymorphic pointers : ");
inh_poly_2::DynamicBase *b_ptr = new inh_poly_2::DynamicDerived;

fmt::println("Type of b_ptr : {}", typeid(b_ptr).name());// static type

// ATTENTION :
//  For pointers you have to dereference to see the dynamic type //
fmt::println("Type of *b_ptr : {}", typeid(*b_ptr).name());
```

```cpp
// typeid can resolve dynamic types
// It's very useful for debugging. In other cases, virtual functions and dynamic_cast are probably better tools.
```