

# Lifetime of the C++ object?

---

Dawid Pilarski

[dawid.pilarski@tomtom.com](mailto:dawid.pilarski@tomtom.com)

[dawid.pilarski@panicsoftware.com](mailto:dawid.pilarski@panicsoftware.com)

[blog.panicsoftware.com](http://blog.panicsoftware.com)

# Introduction

---

# Agenda

---

What does the title mean?

# Who am I?

---

# Questions.

---

Questions...

# What we talk about are basics.

## 6 Basics

[basic]

### 6.7 Memory and objects

[basic.memobj]

#### 6.7.2 Object model

[intro.object]

<sup>1</sup> The constructs in a C++ program create, destroy, refer to, access, and manipulate objects. An *object* is created by a *definition*, by a *new-expression*, when implicitly changing the active member of a *union*, or when a temporary object is created ([conv.rval], [class.temporary]). An object occupies a region of storage in its period of construction ([class.ctor]), throughout its *lifetime*, and in its period of destruction ([class.ctor]). [ *Note*: A function is not an object, regardless of whether or not it occupies storage in the way that objects do. — *end note* ] The properties of an object are determined when the object is created. An object can have a name ([basic.pre]). An object has a storage duration ([basic.stc]) which influences its lifetime ([basic.life]). An object has a type ([basic.types]). Some objects are polymorphic ([class.virtual]); the implementation generates information associated with each such object that makes it possible to determine that object's type during program execution. For other objects, the interpretation of the values found therein is determined by the type of the *expressions* ([expr.compound]) used to access them.

<sup>2</sup> Objects can contain other objects, called *subobjects*. A subobject can be a *member subobject* ([class.mem]), a *base class subobject* ([class.derived]), or an array element. An object that is not a subobject of any other object is called a *complete object*. If an object is created in storage associated with a member subobject or array element *e* (which may or may not be within its lifetime), the created object is a subobject of *e*'s containing object if:

- (2.1) — the lifetime of *e*'s containing object has begun and not ended, and
- (2.2) — the storage for the new object exactly overlays the storage location associated with *e*, and
- (2.3) — the new object is of the same type as *e* (ignoring cv-qualification).

<sup>3</sup> If a complete object is created ([expr.new]) in storage associated with another object *e* of type "array of *N* *unsigned char*" or of type "array of *N* *std::byte*" ([cstddef.syn]), that array *provides storage* for the created object if:

- (3.1) — the lifetime of *e* has begun and not ended, and
- (3.2) — the storage for the new object fits entirely within *e*, and

**What does the title mean?**

---

# Title decomposition

---

What's the lifetime of your object?



# Title decomposition

---

What's the **lifetime** of your object?

- What is a lifetime?

# Title decomposition

---

What's the **lifetime** of your **object**?

- What is a lifetime?
- What is an object?

# Objects

---

# The object

---

Objects are:

- created
- destroyed
- referred to
- accessed
- manipulated

# The Object

---

Is created:

- by the definition

```
int a;
```

# The Object

---

Is created:

- by the definition
- by the new expression

```
new int(5);
```

# The Object

---

Is created:

- by the definition
- by the new expression
- when changing active member of a union

```
union U{int x; int y};
```

```
U u; // active member x;
```

```
u.y = 2; // active member y;
```

# The Object

---

Is created:

- by the definition
- by the new expression
- when changing active member of a union
- by creation of the temporary

```
int{};
```



# The object

---

Has:

- optional name

# The object

---

Has:

- optional name
- storage and it's duration

# The object

---

Has:

- optional name
- storage and it's duration
- lifetime

# The object

---

Has:

- optional name
- storage and it's duration
- lifetime
- type

# The object

---

Is not a reference (although reference has lifetime)

# The variable

---

Can be either an object or the reference.

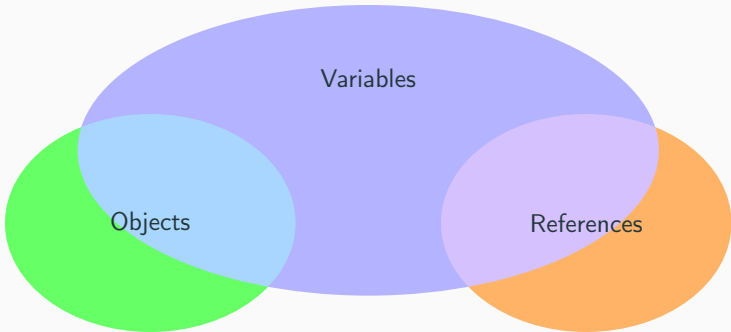
# The variable

---

Can be either an object or the reference. Is introduced by a declaration.

## Summary: variable, reference, object

---





## Summary: variable, reference, object

---

Just in case you want to check validity with the [cppreference](#):

- The object has been recently updated.

## Summary: variable, reference, object

---

Just in case you want to check validity with the [cppreference](#):

- The object has been recently updated.
- The [variable definition](#) is unmaintained and unsupported.

## Summary: variable, reference, object

---

Just in case you want to check validity with the [cppreference](#):

- The object has been recently updated.
- The variable definition is unmaintained and unsupported.
- [Same about references...](#)

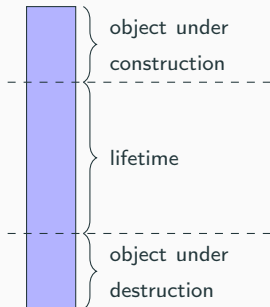
# Lifetime

---

# What is a lifetime?

---

Lifetime is a **runtime** property of an object.



# What is lifetime?

---

During the lifetime of an object you can use it without additional restrictions.

# When the lifetime starts

---

The lifetime of an object starts, when:

- storage with the proper alignment and size for type T is obtained

# When the lifetime starts

---

The lifetime of an object starts, when:

- storage with the proper alignment and size for type T is obtained
- its initialization (if any) is complete



# When the lifetime starts

---

The lifetime of an object starts, when:

- storage with the proper alignment and size for type T is obtained
- its initialization (if any) is complete
- if the object is a union member or subobject thereof, its lifetime only begins if that union member is the initialized member

## When you object does not need initialization

---