# J15 Exceptions

Java Exceptions
throw
Catch or Specify

# Exceptions

- Exceptions are a **control flow construct for error management**.
- Exceptions are used in exceptional/error situations (unlike events, which are expected):
  - A file is not found or is inaccessible.
  - An array is indexed incorrectly (out of bounds).
  - Division by zero.
  - Attempt to access a field or method of a null object.
  - etc.

# Exceptions in Java

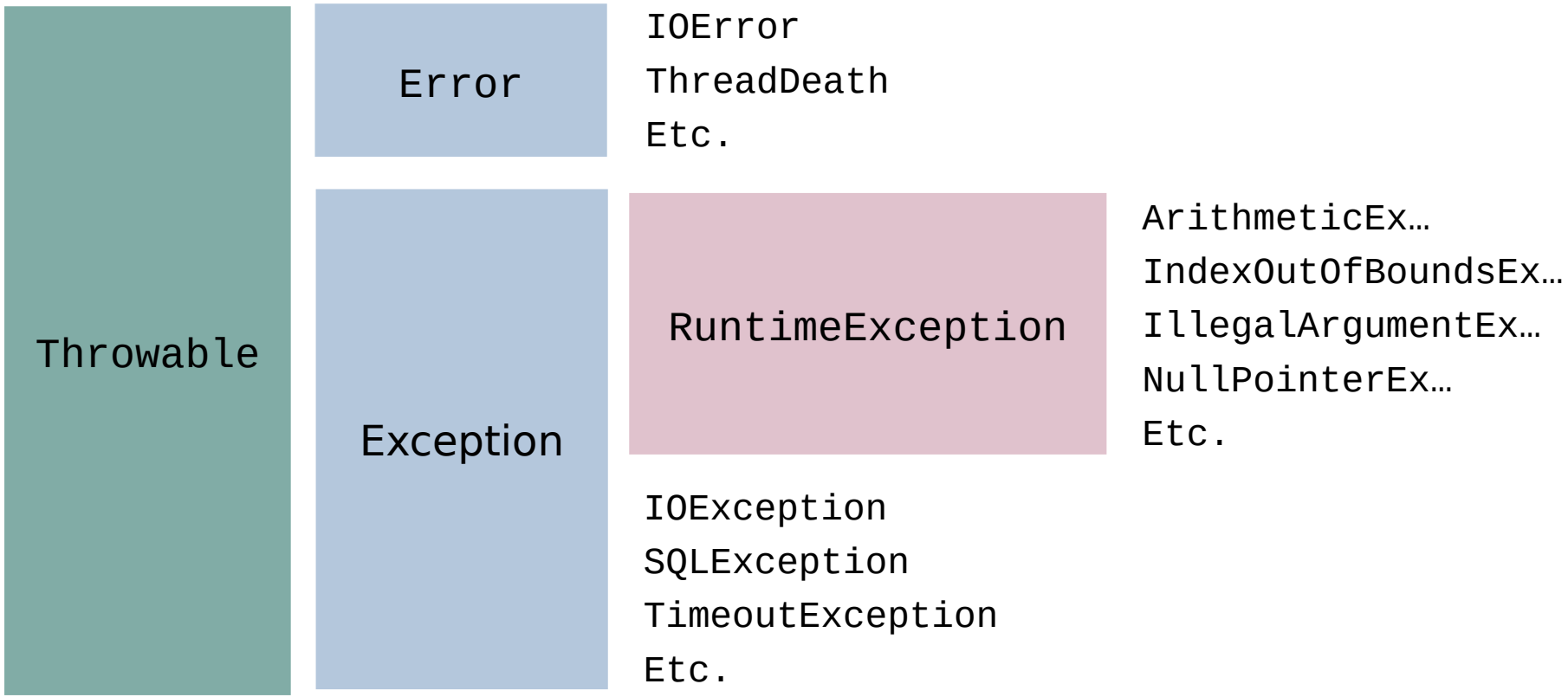
- Exceptions are *thrown* either:
  - Implicitly (via a program error) or
  - Explicitly (by executing the `throw` statement).
- Exceptions may be *caught* with a `try`/`catch` block.
- Exceptions are propagated from callee to caller (call stack is *unwound*) until a matching handler is found.

# Kinds of Java exceptions and compile-time check

- **errors** (`Error` and its subclasses).
  - Serious problems that an application probably shouldn't catch.
- **runtime exceptions** (`RuntimeException` and its subclasses).
  - Exceptional situation that typically cannot be anticipated or recovered from (e.g., program bug, logic error, API misuse): fix the bug rather than catch.
- **checked exceptions** (everything else).
  - Can be thrown during normal operation and can be reasonably anticipated and handled.
- Code that may throw a checked exception must comply with the **catch or specify** requirement, i.e. must be enclosed by either:
  - a `try` statement with a suitable handler, or
  - a method that declares that it `throws` the exception

unchecked exceptions

# Kinds of Java exceptions



# The `throw` statement

- Causes an exception to be thrown:

```
throw new IllegalArgumentException();
```

  - Can throw an instance of any class that extends `Throwable`.
- Why throw exceptions?
  - In a situation where a method cannot do the right thing (what it is supposed to), it is better to signal the error than to silently do the wrong thing (like return a nonsense value).
  - The *fail fast* principle.
- We can define new exception types (subclasses).

# Java `try`/`catch` Block Syntax

```
try {
  // do something that may generate an exception
} catch (ArithmeticException e1) {  // first catch
  // this is an arithmetic exception handler
  // handle the error and/or throw an exception
} catch (Exception e2) {  // may have many catch blocks
  // this an generic exception handler
  // handle the error and/or throw an exception
} finally {
  // this code is guaranteed to run
  // if you need to clean up, put the code here
}
```

- Never catch an exception that you cannot sensibly handle!