The KScr Language

Definition of the KScr Language

comroid

The KScr Language as defined by this document is a compiled high-level language whose goal is to provide runtime optimizations from a different approach.

Team:
Kaleidox

Contents

1 Source Model

1.1 Modifiers

1.1.1 Accessibility Modifiers

Modifier	Function
`public`	Accessible from everywhere
`internal`	Accessible during compilation; compiled to `private`
`protected`	Accessible from inheritors
`private`	Accessible from inside only

1.1.2 Other Modifiers

Modifier	Function
`static`	Cannot be invoked dynamically
`final`	Cannot be overridden or changed
`abstract`	Must be implemented in inheritors
`synchronized`	Invocations are synchronized
`native`	Must be implemented by a native module

1.2 Classes

1.2.1 Class Types

A source class can be of different types:

1.2.1.1 `class`-Type A normal class that can be instantiated. Allowed modifiers are:

Modifier	Function
`static`	Cannot be instantiated and behaves like a Singleton
`final`	Cannot be inherited by other classes
`abstract`	Cannot be instantiated directly and must be inherited by other classes

1.2.1.2 `enum`-Type An enumeration of runtime-constants that follow a class-like pattern. This type does not allow modifiers.

1.2.1.3 `interface`-Type An interface that declares basic structure requirements for implementing classes. Cannot be instantiated directly.

1.2.1.4 `annotation`-Type

1.2.2 Type Generics

1.3 Class Members

1.3.1 Static Initializer

1.3.2 Constructors

1.3.3 Methods

1.3.4 Properties

2 The KScr VM

2.1 Built-in Types

2.1.1 `interface void`

The universal base type. Is implemented by all built-in types and can be implicitly cast to everything.

2.1.2 `class num<T>`

The base type of all numerics. Contains numeric subtypes:

1. `byte` - Type-alias for `int<8>`
2. `short` - Type-alias for `int<16>`
3. `int<n = 32>`
4. `long` - Type-alias for `int<64>`
5. `float`
6. `double`

All subtypes can be used directly, or using the Type Generic T, for example: `int<24> == num<int<24»`

2.1.3 `class str`

2.1.4 `class object`

2.1.5 `class array<T>`

2.1.6 `class tuple<T...>`

2.1.7 `class enum<T>`

2.1.8 `class type<T>`

2.1.9 `class pipe<T...>`

2.2 Literals

2.2.1 `null`

The `null`-Literal. Is always of type `void`.

2.2.2 `Numeric Literals`

2.2.3 `String Literals`

A string literal is pre- and superceded by a double-quote `"` symbol. An escaped double-quote `\"` can be contained in the string.

2.2.3.1 Planned Behaviour: Interpolation A string supports interpolation using accolades with Formatter-support with the following syntax:

```
int hex = 1 << 3;
stdio <<- "hex: {hex:X}"
// prints "hex: 0x4"
```

2.2.4 `Array Literals`

Unimplemented.

2.2.5 `Tuple Literals`

Unimplemented.

2.2.6 `Other Literals`

2.2.6.1 `stdio` Constantly represents the program's standard IO stream.
The held value is of type `pipe<str>`