jminusminus

Class Type

- java.lang.Object
  - jminusminus.Type

- Direct Known Subclasses:
  ArrayTypeName, TypeName

```
class Type
extends Object
```

For representing j-- types. All types are represented underneath (in the classRep field) by Java objects of type Class. These ojects represent types in Java, so this should ease our interfacing with existing Java classes. Class types (reference types that are represented by the identifiers introduced in class declarations) are represented using TypeName. So for now, every TypeName represents a class. In the future, TypeName could be extended to represent interfaces or enumerations. IdentifierTypes must be "resolved" at some point, so that all Types having the same name refer to the same Type object. resolve() does this.

- **Field Summary**

Fields

Modifier and Type	Field and Description
`static Type`	**ANY** The "any" type (denotes wild expressions).
`static Type`	**BOOLEAN** The primitive type, boolean.
`static Type`	**BOXED_BOOLEAN** java.lang.Boolean.

static Type	**BOXED_CHAR** java.lang.Character.
static Type	**BOXED_INT** java.lang.Integer.
static Type	**CHAR** The primitive type, char.
static Type	**CONSTRUCTOR** A type marker indicating a constructor (having no return type).
static Type	**INT** The primitive type, int.
static Type	**NULLTYPE** The null void.
static Type	**OBJECT** The type java.lang.Object.
static Type	**STRING** The type java.lang.String.
static Type	**VOID** The void type.

- **Constructor Summary**

Constructors

Modifier	Constructor and Description
protected	**Type**() This constructor is to keep the compiler happy.

- **Method Summary**

Methods

Modifier and Type	Method and Description
ArrayList<Method>	**abstractMethods**() Return a list of this class' abstract methods? It does has abstract methods if (1) Any method declared in the class is abstract, or (2) Its superclass has an abstract method which is not overridden here.
static String	**argTypesAsString**(Type[] argTypes) Convert an array of argument types to a string representation of a parenthesized list of the types, eg, (int, boolean, java.lang.String).
static boolean	**argTypesMatch**(Class<?>[] argTypes1, Class<?>[] argTypes2) Do argument types match? A helper used for finding candidate methods and constructors.
String	**argumentTypeForAppend**()

	The String representation for a type being appended to a StringBuffer for + and += over strings.
`static boolean`	**`checkAccess`**`(int line, Class referencingType, Class type)` Check the accessibility of a type.
`boolean`	**`checkAccess`**`(int line, Member member)` Check the accessibility of a member from this type (that is, this type is the referencing type).
`boolean`	**`checkAccess`**`(int line, Type targetType)` Check the accesibility of a target type (from this type)
`Class<?>`	**`classRep`**`()` Return the class representation for a type, appropriate for dealing with the Java reflection API.
`Type`	**`componentType`**`()` An array type's component type.
`Constructor`	**`constructorFor`**`(Type[] argTypes)` Find an appropriate constructor in this type, given it's argument types.
`boolean`	**`equals`**`(Type that)` Type equality is based on the equality of descriptors.
`Field`	**`fieldFor`**`(String name)` Return the Field having this name.
`boolean`	**`isAbstract`**`()` Is this type declared abstract?
`boolean`	**`isArray`**`()` Is this an Array type?
`boolean`	**`isFinal`**`()` Is this type declared final?
`boolean`	**`isInterface`**`()` Is this an interface type?
`boolean`	**`isJavaAssignableFrom`**`(Type that)` Is this a supertype of that?
`boolean`	**`isPrimitive`**`()` Is this a primitive type?
`boolean`	**`isReference`**`()` Is this a reference type?
`String`	**`jvmName`**`()` The JVM representation for this type's name.
`boolean`	**`matchesExpected`**`(Type expected)` Does this type match the expected type? For now, "matches" means "equals".
`Method`	**`methodFor`**`(String name, Type[] argTypes)` Find an appropriate method in this type, given a message (method) name and it's argument types.

`void`	`mustMatchExpected(int line, Type expectedType)` An assertion that this type matches the specified type.
`void`	`mustMatchOneOf(int line, Type... expectedTypes)` An assertion that this type matches one of the specified types.
`String`	`packageName()` Return the type's package name.
`Type`	`resolve(Context context)` Resolve this type in the given context.
`void`	`setClassRep(Class<?> classRep)` This setter is used by JCompilationUnit.preAnalyze() to set the classRep to the specified partial class, computed during pre-analysis.
`static String`	`signatureFor(String name, Type[] argTypes)` A helper for constructing method signatures for reporting unfound methods and constructors.
`String`	`simpleName()` Return the simple (unqualified) name for this Type.
`Type`	`superClass()` Return the Type's super type (or null if there is none).
`String`	`toDescriptor()` The JVM descriptor for this type.
`String`	`toString()` A printable (j--) string representation of this type.
`static Type`	`typeFor(Class<?> classRep)` Constructs a Type representation for a type from its (Java) Class representation.

- **Methods inherited from class java.lang.Object**

  `clone, equals, finalize, getClass, hashCode, notify, notifyAll, wait, wait, wait`

- **Field Detail**

  - **INT**

  `public static finalType INT`

  The primitive type, int.

  - **CHAR**

  `public static finalType CHAR`

  The primitive type, char.

  - **BOOLEAN**

  `public static finalType BOOLEAN`

  The primitive type, boolean.

- **BOXED_INT**

```
public static finalType BOXED_INT
```

java.lang.Integer.

- **BOXED_CHAR**

```
public static finalType BOXED_CHAR
```

java.lang.Character.

- **BOXED_BOOLEAN**

```
public static finalType BOXED_BOOLEAN
```

java.lang.Boolean.

- **STRING**

```
public staticType STRING
```

The type java.lang.String.

- **OBJECT**

```
public staticType OBJECT
```

The type java.lang.Object.

- **VOID**

```
public static finalType VOID
```

The void type.

- **NULLTYPE**

```
public static finalType NULLTYPE
```

The null void.

- **CONSTRUCTOR**

```
public static finalType CONSTRUCTOR
```

A type marker indicating a constructor (having no return type).

- **ANY**

```
public static finalType ANY
```

The "any" type (denotes wild expressions).

- **Constructor Detail**

  - **Type**

```
protectedType()
```

This constructor is to keep the compiler happy.

- **Method Detail**

  - **typeFor**

```
public staticTypetypeFor(Class<?>classRep)
```

Construct a Type representation for a type from its (Java) Class representation. Make sure there is a unique Type for each unique type.

Parameters:

`classRep` - the Java representation.

- **classRep**

```
publicClass<?>classRep()
```

Return the class representation for a type, appropriate for dealing with the Java reflection API.

Returns:

the Class representation for this type.

- **setClassRep**

```
publicvoidsetClassRep(Class<?>classRep)
```

This setter is used by JCompilationUnit.preAnalyze() to set the classRep to the specified partial class, computed during pre-analysis.

Parameters:

`classRep` - the partial class.

- **equals**

```
publicbooleanequals(Typethat)
```

Type equality is based on the equality of descriptors.

Parameters:

`that` - the other Type.

Returns:

true iff the two types are equal.

- **isArray**

```
publicbooleanisArray()
```

Is this an Array type?

Returns:

true or false.

- **componentType**

```
publicTypecomponentType()
```

An array type's component type. Meaningful only for array types.

Returns:

the component type.

- **superClass**

```
publicTypesuperClass()
```

Return the Type's super type (or null if there is none). Meaningful only to class Types.

Returns:

the super type.

- **isPrimitive**

```
publicbooleanisPrimitive()
```

Is this a primitive type?

Returns:

true or false.

- **isInterface**

```
publicbooleanisInterface()
```

Is this an interface type?

Returns:

true or false.

- **isReference**

```
publicbooleanisReference()
```

Is this a reference type?

Returns:

true or false.

- **isFinal**

```
publicbooleanisFinal()
```

Is this type declared final?

Returns:

true or false.

- **isAbstract**

```
publicbooleanisAbstract()
```

Is this type declared abstract?

Returns:

true or false.

- **isJavaAssignableFrom**

```
publicbooleanisJavaAssignableFrom(Typethat)
```

Is this a supertype of that?

Parameters:

`that` - the candidate subtype.

Returns:

true iff this is a supertype of that.

- **abstractMethods**

```
publicArrayList<Method>abstractMethods()
```

Return a list of this class' abstract methods? It does has abstract methods if (1) Any method declared in the class is abstract, or (2) Its superclass has an abstract method which is not overridden here.

Returns:

a list of abstract methods.

- **mustMatchOneOf**

```
publicvoidmustMatchOneOf(intline,
                         Type...expectedTypes)
```

An assertion that this type matches one of the specified types. If there is no match, an error message is returned.

Parameters:

`line` - the line near which the mismatch occurs.
`expectedTypes` - expected types.

- **mustMatchExpected**

```
publicvoidmustMatchExpected(intline,
                            TypeexpectedType)
```

An assertion that this type matches the specified type. If there is no match, an error message is written.

Parameters:
`line` - the line near which the mismatch occurs.
`expectedType` - type with which to match.

- **matchesExpected**

```
publicbooleanmatchesExpected(Typeexpected)
```

Does this type match the expected type? For now, "matches" means "equals".

Parameters:
`expected` - the type that this might match.

Returns:
true or false.

- **argTypesMatch**

```
public static booleanargTypesMatch(Class<?>[]argTypes1,
                          Class<?>[]argTypes2)
```

Do argument types match? A helper used for finding candidate methods and constructors.

Parameters:
`argTypes1` - arguments (classReps) of one method.
`argTypes2` - arguments (classReps) of another method.

Returns:
true iff all corresponding types of argTypes1 and argTypes2 match.

- **simpleName**

```
publicStringsimpleName()
```

Return the simple (unqualified) name for this Type. Eg, String in place of java.lang.String.

Returns:
the simple name.

- **toString**

```
publicStringtoString()
```

A printable (j--) string representation of this type. Eg, int[], java.lang.String.

Overrides:
`toString` in class `Object`

Returns:
the string representation.

- **toDescriptor**

```
publicStringtoDescriptor()
```

The JVM descriptor for this type. Eg, Ljava/lang/String; for java.lang.String, [[Z for boolean[][].

Returns:
the descriptor.

- **jvmName**

```
publicStringjvmName()
```

The JVM representation for this type's name. This is also called the internal form of the name. Eg, java/lang/String for java.lang.String.

Returns:
the type's name in internal form.

- **packageName**

```
publicStringpackageName()
```

Return the type's package name. Eg, java.lang for java.lang.String.

Returns:
the package name.

- **argumentTypeForAppend**

```
publicStringargumentTypeForAppend()
```

The String representation for a type being appended to a StringBuffer for + and += over strings.

Returns:
a string representation of the type.

- **methodFor**

```
publicMethodmethodFor(Stringname,
                      Type[]argTypes)
```

Find an appropriate method in this type, given a message (method) name and it's argument types. This is pretty easy given our (current) restriction that the types of the actual arguments must exactly match the types of the formal parameters. Returns null if it cannot find one.

Parameters:
`name` - the method name.
`argTypes` - the argument types.

Returns:
Method with given name and argument types, or null.

- **constructorFor**

```
publicConstructorconstructorFor(Type[]argTypes)
```

Find an appropriate constructor in this type, given it's argument types. This is pretty easy given our (current) restriction that the types of the actual arguments must exactly match the types of the formal parameters. Returns null if it cannot find one.

Parameters:
`argTypes` - the argument types.

Returns:
Constructor with the specified argument types, or null.

- **fieldFor**

```
publicFieldfieldFor(Stringname)
```

Return the Field having this name.

Parameters:

`name` - the name of the field we want.

Returns:

the Field or null if it's not there.

- **argTypesAsString**

```
public staticStringargTypesAsString(Type[]argTypes)
```

Convert an array of argument types to a string representation of a parenthesized list of the types, eg, (int, boolean, java.lang.String).

Parameters:

`argTypes` - the array of argument types.

Returns:

the string representation.

- **checkAccess**

```
publicbooleancheckAccess(intline,
                         Membermember)
```

Check the accessibility of a member from this type (that is, this type is the referencing type).

Parameters:

`line` - the line in which the access occurs.
`member` - the member being accessed.

Returns:

true if access is valid; false otherwise.

- **checkAccess**

```
publicbooleancheckAccess(intline,
                         TypetargetType)
```

Check the accessibility of a target type (from this type)

Parameters:

`line` - line in which the access occurs.
`targetType` - the type being accessed.

Returns:

true if access is valid; false otherwise.

- **checkAccess**

```
public staticbooleancheckAccess(intline,
                         ClassreferencingType,
                         Classtype)
```

Check the accessibility of a type.

Parameters:

`line` - the line in which the access occurs.
`referencingType` - the type attempting the access.
`type` - the type that we want to access.

Returns:

true if access is valid; false otherwise.

- **resolve**

```
publicTyperesolve(Contextcontext)
```

Resolve this type in the given context. Notice that this has meaning only for TypeName and ArrayTypeName, where names are replaced by real types. Names are looked up in the context.

Parameters:

`context` - context in which the names are resolved.

Returns:

the resolved type.

- **signatureFor**

```
public staticStringsignatureFor(Stringname,
                              Type[]argTypes)
```

A helper for constructing method signatures for reporting unfound methods and constructors.

Parameters:

`name` - the message or Type name.

`argTypes` - the actual argument types.

Returns:

a printable signature.