Reflection

Contents

1. Getting started with reflection
2. Accessing metadata information
3. Creating and using objects

Demo project: **DemoReflection**

2

1. Getting Started with Reflection

- Recap of classes
- Contents of a .class file
- Accessing metadata at run-time

Recap of Classes

- The fully qualified name for a class/interface includes its package name
  - Outer class/interface: `package.Type`
  - Inner class/interface: `package.OuterType.InnerType`
- Each Java class/interface is compiled into a separate `.class` file, with the following name:
  - Outer class/interface: `Type.class`
  - Inner class/interface: `OuterType$InnerType.class`

4

Contents of a .class File

- `.class` files contain the following information:
  - Byte codes for all the Java code
  - Constants (string literals and numbers)
  - Metadata about the class/interface housed in the `.class` file
- Metadata information available:
  - Is the type a class or interface?
  - Modifiers on the type (e.g. is it public, is it abstract, etc.)
  - Name of the super-type
  - List of interfaces implemented
  - Full metadata about all method/constructor signatures
  - Full metadata about all fields

5

Accessing Metadata at Run-Time

- You can access class/interface metadata at run-time
  - Via the Java Reflection API
- This enables you to write extremely adaptable code
  - You can discover the full capabilities of a type at run-time
  - You can then create instances of a type, invoke its methods, and access its fields
- You can do all of this, without any compile-time dependency on the class/interface type
  - Allows you to plug in completely new classes/interfaces after deployment
  - Extreme flexibility!

6

2. Accessing Metadata Information

- Getting class information
- Getting method information
- Getting constructor information
- Getting field information

7

Getting Class Info

- Full information about a class/interface is provided via the `Class<?>` type
- There are several ways to get a `Class<?>` for an existing object:

```
Class<?> classInfo = anObject.getClass();
```

```
Class<?> classInfo = Class.forName("fully-qualified-typename");
```

```
Class<?> classInfo = TypeName.class;
```

8

Getting Class Info – Example

- This example shows how to get class information for some existing objects

```
public class DemoClassInfo {

  public static void main(String[] args) {

    String name = "John";
    Date now = new Date();
    int num = 42;

    // Get a Class<?> via getClass(), can also use Class.forName() or ClassName.class
    displayClassInfo(name.getClass());
    displayClassInfo(now.getClass());
    displayClassInfo(new Integer(num).getClass());
    displayClassInfo(args.getClass());
  }


  private static void displayClassInfo(Class<?> classInfo) {
    System.out.printf("Class name:  %s\n",   classInfo.getName());
    System.out.printf("Simple name: %s\n\n", classInfo.getSimpleName());
  }
}
```

Getting Method Info

- `Class<?>` allows you to get info about methods in a type
  - `getDeclaredMethods()`
    - Returns `Method[]` of all methods declared in this type (excludes inherited)
  - `getMethods()`
    - Returns `Method[]` of public methods available in this type (includes inherited)
- `java.lang.reflect.Method` describes:
  - Method name
    - Expressed as a `String`
  - Return type, parameter types, and exception types
    - Type info is expressed as `Class<?>`
  - Method modifiers
    - Expressed as an integer (use the `Modifier` class to process the info)
    - Methods, constructors, and fields all have modifier info

10

Getting Method Info – Example (1)

- Get method information for a specified type:

```
public class DemoMethodInfo {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a fully qualified Java typename: ");
    String typeName = scanner.nextLine();

    try {
      Method[] methods = Class.forName(typeName).getDeclaredMethods();
      for (Method method: methods) {
        displayMethodInfo(method);
        displayModifierInfo(method);
      }
    }
    catch (ClassNotFoundException ex) {
      System.out.println("Exception: class not found");
    }
  }

  // See displayMethodInfo() and displayModifierInfo() on the next slide.
  …
}
```

Getting Method Info – Example (2)

- Example continued:

```
private static void displayMethodInfo(Method method) {
  System.out.printf("Method name: %s\n", method.getName());
  System.out.printf("Return type: %s\n", method.getReturnType().getName());
  System.out.printf("Parameters:  %d\n", method.getParameterTypes().length);
  System.out.printf("Exceptions:  %d\n", method.getExceptionTypes().length);
}
```

```
private static void displayModifierInfo(Method method) {

  int mod = method.getModifiers();
  StringBuffer buf = new StringBuffer();

  if (Modifier.isPublic(mod))
    buf.append("public ");
  else if (Modifier.isPrivate(mod))
    buf.append("private ");
  else if (Modifier.isProtected(mod))
    buf.append("protected ");

  if (Modifier.isAbstract(mod))
    buf.append("abstract ");
  else if (Modifier.isStatic(mod))
    buf.append("static");

  System.out.printf("Modifiers:   %s\n\n", buf.toString());
}
```

12

Getting Constructor Info

- `Class<?>` allows you to get info about constructors in a type
  - `getDeclaredConstructors()`
    - Returns `Constructor<?>[]` of all constructors for this type
  - `getConstructors()`
    - Returns `Constructor<?>[]` of public constructors for this type
- `java.lang.reflect.Constructor` describes:
  - Constructor name
    - Expressed as a `String`
  - Parameter types and exception types
    - Type info is expressed as `Class<?>`
  - Constructor modifiers
    - Expressed as an integer (use the `Modifier` class to process the info)
    - Methods, constructors, and fields all have modifier info

13

Getting Constructor Info – Example (1)

- Get constructor information for a specified type:

```
public class DemoConstructorInfo {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a fully qualified Java typename: ");
    String typeName = scanner.nextLine();

    try {
      Constructor<?>[] ctors = Class.forName(typeName).getDeclaredConstructors();
      for (Constructor<?> ctor: ctors) {
        displayConstructorInfo(ctor);
        displayModifierInfo(ctor);
      }
    }
    catch (ClassNotFoundException ex) {
      System.out.println("Exception: class not found");
    }
  }

  // See displayConstructorInfo() and displayModifierInfo() on the next slide.
  …
}
```

Getting Constructor Info – Example (2)

- Example continued:

```
private static void displayConstructorInfo(Constructor<?> ctor) {
  System.out.printf("Constructor name: %s\n", ctor.getName());
  System.out.printf("Parameters:  %d\n", ctor.getParameterTypes().length);
  System.out.printf("Exceptions:  %d\n", ctor.getExceptionTypes().length);
}
```

```
private static void displayModifierInfo(Constructor<?> ctor) {
  int mod = ctor.getModifiers();
  StringBuffer buf = new StringBuffer();

  if (Modifier.isPublic(mod))
    buf.append("public ");
  else if (Modifier.isPrivate(mod))
    buf.append("private ");
  else if (Modifier.isProtected(mod))
    buf.append("protected ");

  System.out.printf("Modifiers:   %s\n\n",  buf.toString());
}
```

Getting Field Info

- `Class<?>` allows you to get info about fields in a type
  - `getDeclaredFields()`
    - Returns `Field[]` of all fields declared in this type (excludes inherited)
  - `getFields()`
    - Returns `Field[]` of public fields available in this type (includes inherited)
- `java.lang.reflect.Field` describes:
  - Field name
    - Expressed as a `String`
  - Field type
    - Type info is expressed as `Class<?>`
  - Field modifiers
    - Expressed as an integer (use the `Modifier` class to process the info)
    - Methods, constructors, and fields all have modifier info

16

Getting Field Info – Example (1)

- Get field information for a specified type:

```
public class DemoFieldInfo {

  public static void main(String[] args) {

    Scanner scanner = new Scanner(System.in);
    System.out.print("Enter a fully qualified Java typename: ");
    String typeName = scanner.nextLine();

    try {
      Field[] fields = Class.forName(typeName).getDeclaredFields();
      for (Field field: fields) {
        displayFieldInfo(field);
        displayModifierInfo(field);
      }
    }
    catch (ClassNotFoundException ex) {
      System.out.println("Exception: class not found");
    }
  }

  // See displayFieldInfo() and displayModifierInfo() on the next slide.
  …
}
```

Getting Field Info – Example (2)

- Example continued:

```
private static void displayFieldInfo(Field field) {
  System.out.printf("Field name: %s\n", field.getName());
  System.out.printf("Field type: %s\n", field.getType());
}
```

```
// This method is similar to before :-)

private static void displayModifierInfo(Field field) {

  int mod = field.getModifiers();
  StringBuffer buf = new StringBuffer();

  if (Modifier.isPublic(mod))
    buf.append("public ");
  else if (Modifier.isPrivate(mod))
    buf.append("private ");
  else if (Modifier.isProtected(mod))
    buf.append("protected ");

  if (Modifier.isStatic(mod))
    buf.append("static");

  System.out.printf("Modifiers: %s\n\n",  buf.toString());
}
```

3. Creating and Using Objects

- Creating an object
- Invoking methods
- Accessing field values

19

Creating an Object

- You can use reflection to create an object
  - Allows you to create objects for types that are not known at compile-time
- How to do it:
  - First, get a `Class<?>` for the required type
    - Via `TypeName.class`
    - Or `Class.forName("typeName")`
  - Then, get a `Constructor<?>` for the type
    - Via `getDeclaredConstructor(`*`parameterTypes`*`)`
    - Or `getConstructor(`*`parameterTypes`*`)`
  - Then create an instance
    - Via `newInstance(`*`parameterValues`*`)`

20

Creating an Object – Example

- Here's a simple class with a constructor

```
public class Book {
  public Book(String title, String author, double price) { … }
  …
}
```

- Here's some code to create an instance via reflection:

```
public class DemoObjectCreation {

  public static void main(String[] args) {
    try {
      Class<Book> bookClass = Book.class;
      Constructor<Book> bookCtor = bookClass.getConstructor(String.class,
                                                            String.class,
                                                            Double.TYPE);

      Book aBook = bookCtor.newInstance("The Gruffalo", "Julia Donaldson", 6.99);
      System.out.println(aBook);
    }
    catch (NoSuchMethodException ex) { … }
    catch (IllegalAccessException ex) { … }
    catch (InstantiationException ex) { … }
    catch (InvocationTargetException ex) { … }
  }
}
```

21

Invoking Methods

- You can use reflection to invoke a method
  - Allows you to make use of objects created via reflection
- How to do it:
  - First, get a `Class<?>` for the required type
    - Via `TypeName.class`
    - Or `Class.forName("typeName")`
  - Then, get a `Method` for the type
    - Via `getDeclaredMethod(`*`parameterTypes`*`)`
    - Or `getMethod(`*`parameterTypes`*`)`
  - Then invoke the method
    - Via `Invoke(`*`targetObject, parameterValues`*`)`

22

Invoking Methods – Example

- Here's a simple class with some methods

```
public class Book {
  public double increasePrice(double amount) { … }
  public String toString() { … }
  …
}
```

- Here's some code to create an instance via reflection:

```
public class DemoMethodInvocation {

  public static void main(String[] args) {
    try {
      Class<Book> bookClass = Book.class;
      Book aBook;
      …
      Method method = bookClass.getMethod("increasePrice", Double.TYPE);
      double newPrice = (Double)method.invoke(aBook, 1.50);

      System.out.printf("New price: %.2f\n", newPrice);
    }
    catch (NoSuchMethodException ex) { … }
    catch (IllegalAccessException ex) { … }
    catch (InstantiationException ex) { … }
    catch (InvocationTargetException ex) { … }
  }
}
```

23

Accessing Field Values

- `java.lang.reflect.Field` allows you to get/set accessible fields on an object
  - First, get a `Field` object to specify an accessible field on a type
  - Then call a getter method to get the field value on a target object

```
Object objectValue = aField.get(targetObject);
int    intValue    = aField.getInt(targetObject);
double doubleValue = aField.getDouble(targetObject);
…
```

  - And/or call a setter method to set the field value on a target object

```
aField.set(targetObject, objectValue);
aField.setInt(targetObject, intValue);
aField.setDouble(targetObject, doubleValue);
…
```

24

Accessing Field Values – Example

- Here's a simple class with some public fields(!)

```
public class Book {
  public String title;
  public String author;
  public double price;   // Etc.
}
```

- Here's some code to get a field value via reflection:

```
public class DemoFieldValues {

  public static void main(String[] args) {

    Book aBook = new Book("Hitch-hikers Guide to the Galaxy", "Douglas Adams", 6.99);

    try {
      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the name of an accessible field in the Book class: ");
      String fieldName = scanner.nextLine();

      Class<Book> bookClass = Book.class;
      Field field = bookClass.getField(fieldName);
      System.out.printf("Value of %s field: %s", fieldName, field.get(aBook));
    }
    catch (NoSuchFieldException ex) { … }
    catch (IllegalAccessException ex) { … }
  }
}
```

25

Any Questions?

26