The Future of Java

Bob Lee

Google Inc.

Who is Bob Lee?

> Google engineer

> Android core library lead

> Guice creator

> JSR-330 lead

> EC rep

> St. Louisan

> Speedo model

Let's talk about...

> Project Coin
> JSR-330: Dependency Injection for Java
> MapMaker

Project Coin

Small language **change**s

Currently accepted proposals

> Strings in switch

> Automatic Resource Management (ARM)

> Improved generic type inference for constructors

> Simplified varags method invocation

> Collection literals and access syntax

> Better integral literals

> JSR-292 (Invokedynamic) support

Currently accepted proposals

> Strings in switch
> **Automatic Resource Management (ARM)**
> Improved generic type inference for constructors
> **Simplified varags method invocation**
> Collection literals and access syntax
> Better integral literals
> JSR-292 (Invokedynamic) support

ARM

- Automatic Resource Management
- Helps dispose of resources
- Proposed by Josh Bloch

Example: Parsing a file header

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    Header header = parse(in.readLine());
    in.close();
    return header;
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

Example: Parsing a file header

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    Header header = parse(in.readLine());
    in.close();
    return header;
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

See the problem?

If we don't reach **`close()`**, we leak.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    Header header = parse(in.readLine());
    in.close();
    return header;
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

finally ensures **close()** is always called.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    try {
      return parse(in.readLine());
    } finally {
      in.close();
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

But what happens when **`close()`** throws?

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    try {
      return parse(in.readLine());
    } finally {
      in.close();
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

We *could* ignore the exception from **`close()`**.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    try {
      return parse(in.readLine());
    } finally {
      try { in.close(); } catch (IOException e) { /* ignore */ }
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

But it's better to throw the right exception.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    BufferedReader in = new BufferedReader(new FileReader(file));
    boolean successful = false;
    try {
      Header header = parse(in.readLine());
      successful = true;
      return header;
    } finally {
      try { in.close(); } catch (IOException e) {
        if (successful) throw e;
        else e.printStackTrace(); // let original exception propagate
      }
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

Equivalent code, using an ARM block.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    try (BufferedReader in = new BufferedReader(
        new FileReader(file))) {
      return parse(in.readLine());
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

Equivalent code, using an ARM block.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    try (BufferedReader in = new BufferedReader(
        new FileReader(file))) {
      return parse(in.readLine());
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

Note: Techincally, we could still leak.

Equivalent code, using an ARM block.

```
public class HeaderParser {
  /** Parses header from the first line of file. */
  public static Header parse(File file) throws IOException,
      ParseException {
    try (Reader fin = new FileReader(file);
        BufferedReader in = new BufferedReader(fin)) {
      return parse(in.readLine());
    }
  }

  private static Header parse(String first) throws ParseException {
    ...
  }
}
```

Why ARM is important

> Out of 110 uses of **`close()`** in the JDK...
  - 74 (2/3) leaked
  - In other words, 2/3 of all uses were broken

>