Interfaces

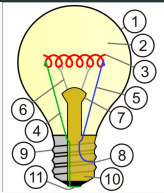
COMP122: Object-Oriented Programming
Patrick Totzke
totzke@liverpool.ac.uk

UNIVERSITY OF LIVERPOOL

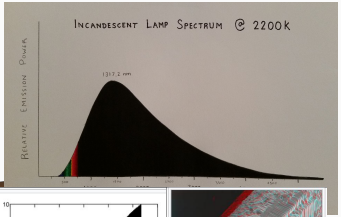
Interfaces

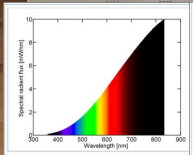
Interfaces

Interfaces

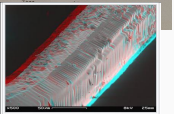


1. Outline of Glass bulb
2. Low pressure inert gas (argon, nitrogen, krypton, xenon)
3. Tungsten filament
4. Contact wire (goes out of stem)
5. Contact wire (goes into stem)
6. Support wires (one end embedded in stem; conduct no current)
7. Stem (glass mount)
8. Contact wire (goes out of stem)
9. Cap (sleeve)
10. Insulation (vitrite)
11. Electrical contact



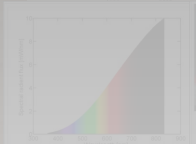


Spectral power distribution of a 25 W incandescent light bulb.



Filament of a 50-watt incandescent lightbulb in an SEM in stereoscopic mode, presented as an anaglyph image. 3D red cyan glasses are recommended to view this image correctly.

Interfaces



Spectral power distribution of a 25 W incandescent light bulb.

Filament of a 50-watt incandescent lightbulb in an SEM in stereoscopic mode, presented as an anaglyph image. 3D red/cyan glasses are recommended to view this image correctly.

Interfaces

```
1  public class Lamp {
2    public void switchOn() {
3      // physics
4    }
5    public void switchOff() {
6      // physics
7    }
8  }
```

Interfaces

```
1  Lamp l = new Lamp();
2  l.switchOn();
3  l.switchOff();
```

```
1  public class Lamp {
2    public void switchOn() {
3      // physics
4    }
5    public void switchOff() {
6      // physics
7    }
8  }
```

Interfaces

```java
public interface Switchable {
  public void switchOn();
  public void switchOff();
}
```

```java
Lamp l = new Lamp();
l.switchOn();
l.switchOff();
```

```java
public class Lamp {
  public void switchOn() {
    // physics
  }
  public void switchOff() {
    // physics
  }
}
```

Interfaces

```java
public interface Switchable {
  public void switchOn();
  public void switchOff();
}
```

```java
Switchable l = new Lamp();
l.switchOn();
l.switchOff();
```

```java
public class Lamp {
  public void switchOn() {
    // physics
  }
  public void switchOff() {
    // physics
  }
}
```

Interfaces

```
1  public interface Switchable {
2    public void switchOn();
3    public void switchOff();
4  }
```

```
1  Switchable l = new Lamp();
2  l.switchOn();
3  l.switchOff();
```

```
1  public class Lamp
2          implements Switchable {
3    public void switchOn() {
4      // physics
5    }
6    public void switchOff() {
7      // physics
8    }}
```

In Java, an interface is a specification which public methods must exist in a class that *implements* it.

- Interfaces define a type,
- (typically) only contain constants and method signatures,
- cannot be instantiated,
- can be implemented by a class, which then has to contain all method bodies declared in the interface,
- can be extended by other interfaces.
- A class can implement more than one interface.

In Java, an interface is a specification which public methods must exist in a class that *implements* it.

- Interfaces define a type,
- (typically) only contain constants and method signatures,
- cannot be instantiated,
- can be implemented by a class, which then has to contain all method bodies declared in the interface,
- can be extended by other interfaces.
- A class can implement more than one interface.

A class can implement several interfaces. Such a class would have to implement all methods from all the interfaces it implements.

```
1 public class Lamp implements Switchable, Pluggable {
2       /* methods from Switchable */
3       /* methods from Plugable */
4 }
```

A class can implement several interfaces. Such a class would have to implement all methods from all the interfaces it implements.

```
1 public class Lamp implements Switchable, Pluggable {
2       /* methods from Switchable */
3       /* methods from Plugable */
4 }
```

An interface can itself `extend` (one or more!) other interfaces

```
1 public interface Dimmable extends Switchable, Plugable {...
```

A class can implement several interfaces. Such a class would have to implement all methods from all the interfaces it implements.

```
1 public class Lamp implements Switchable, Pluggable {
2       /* methods from Switchable */
3       /* methods from Plugable */
4 }
```

An interface can itself `extend` (one or more!) other interfaces

```
1 public interface Dimmable extends Switchable, Plugable {...
```

but cannot `implement` other interfaces.

```
1 public interface Dimmable implements Switchable {
```

A class can implement several interfaces. Such a class would have to implement all methods from all the interfaces it implements.

```
1 public class Lamp extends Furniture
2                   implements Switchable, Pluggable {
3       /* methods from Switchable */
4       /* methods from Plugable */
5 }
```

An interface can itself `extend` (one or more!) other interfaces

```
1 public interface Dimmable extends Switchable, Plugable {...
```

but cannot `implement` other interfaces.

```
1 public interface Dimmable implements Switchable {
```

Conventions

- By convention interface identifiers end in "-able". Examples are `java.lang.Comparable` and `java.io.Serializable`.
- All interface methods are `public` and `abstract`
- All interface attributes are `public`, `static`, and `final`.

Interfaces vs Abstract classes

Abstract Classes	Interfaces
can only `extend` one superclass	multiple inheritance between interfaces
can extend any class	can only extend interfaces
may have `abstract` and concrete methods	only abstract methods
`protected` methods allowed	methods are `public abstract`
no restriction on attribute modifiers	only `public static final` vars