Object Orientated Software Engineering

- 3.2 Class Diagrams, Encapsulation, and Abstraction
- By Stephen Lindsay

University of Glasgow

Outline

1. Inheritance
2. Polymorphism
3. Abstraction of Inheritance in Java?

University of Glasgow

Principle 3: Inheritance

Inheritance is the ability for objects to inherit variable and functional behaviour from a parent object. This allows for substantial code re-use.

The benefits of inheritance are

- Code re-use so we have to write less code
- Extensibility – can extend class logic as needed
- Hiding data – we can keep data private (encapsulation again)

University of Glasgow

Inheritance in Java

```
public class PowerTool
{
  private boolean batteryPowered;
  private Date lastElectricalTest;
  private int operatingLifetime;

  // check safe to use, renew lastElectricalTest
  public void serviceTool()

  // check if power tool needs to be serviced
  public boolean needsService()
```

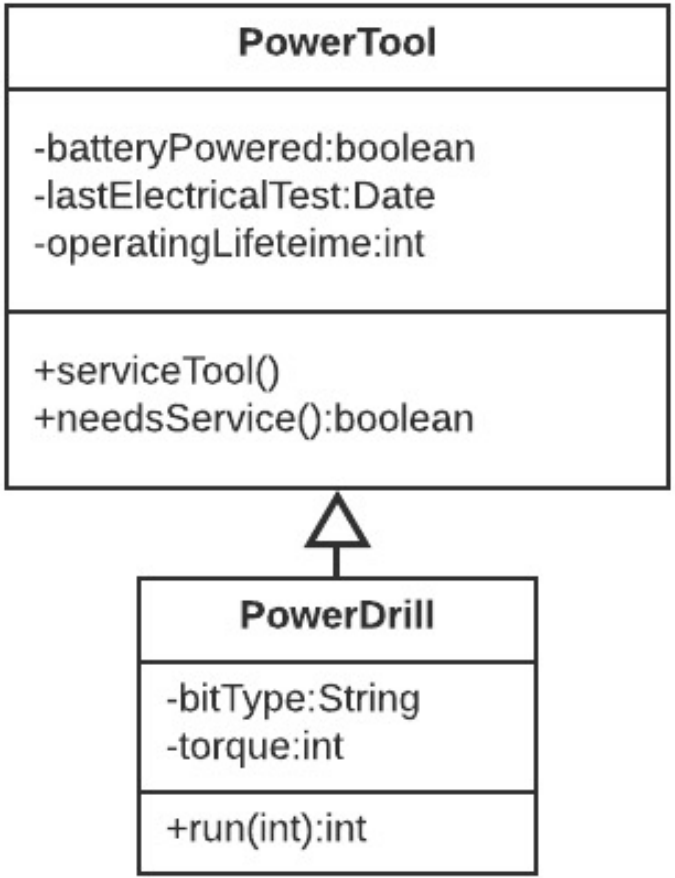
University of Glasgow

Inheritance in Java

```
public class PowerDrill extends PowerTool
{
  private String bitType;
  private int torque;

  private int run(int hours)
  {
    // increment operating lifetime
    // reduce battery life if needed
    // if drilling, run breakageCheck()
    // return work done over this time
  }
}
```

UML and Inheritance



University of Glasgow

Multi-Level Inheritance in Java

Hierarchical inheritance in Java:

```
public class Jigsaw extends PowerTool
public class Drill extends PowerTool
public class CircularSaw extends PowerTool
```

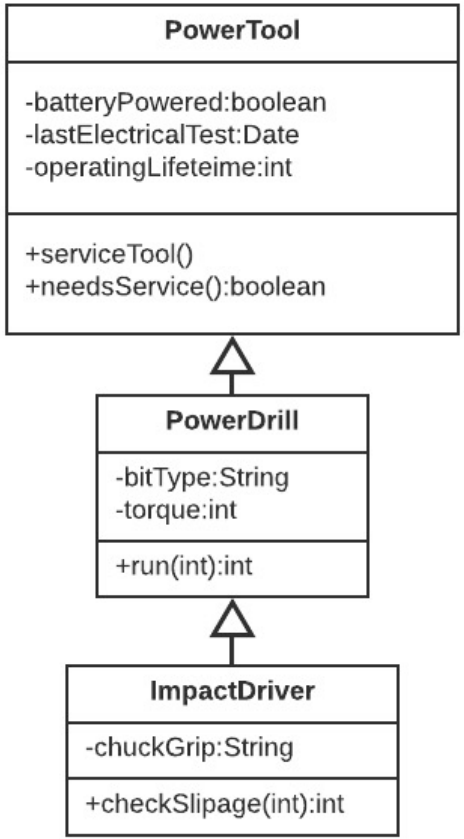
- Allows us to code base methods in one place and maintain code there

Multi-level inheritance in Java:

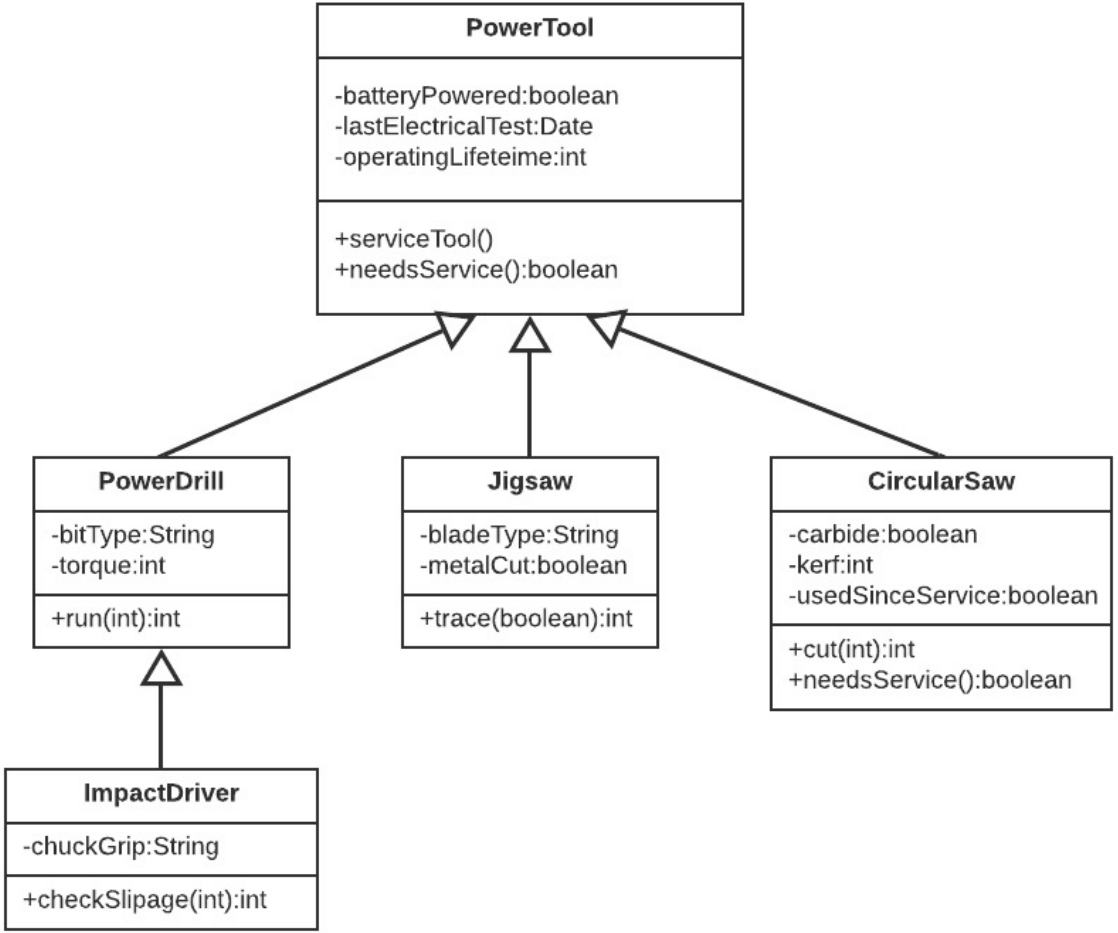
```
public class ImpactDriver extends PowerTool
```

- Build up layers of specialist functionality

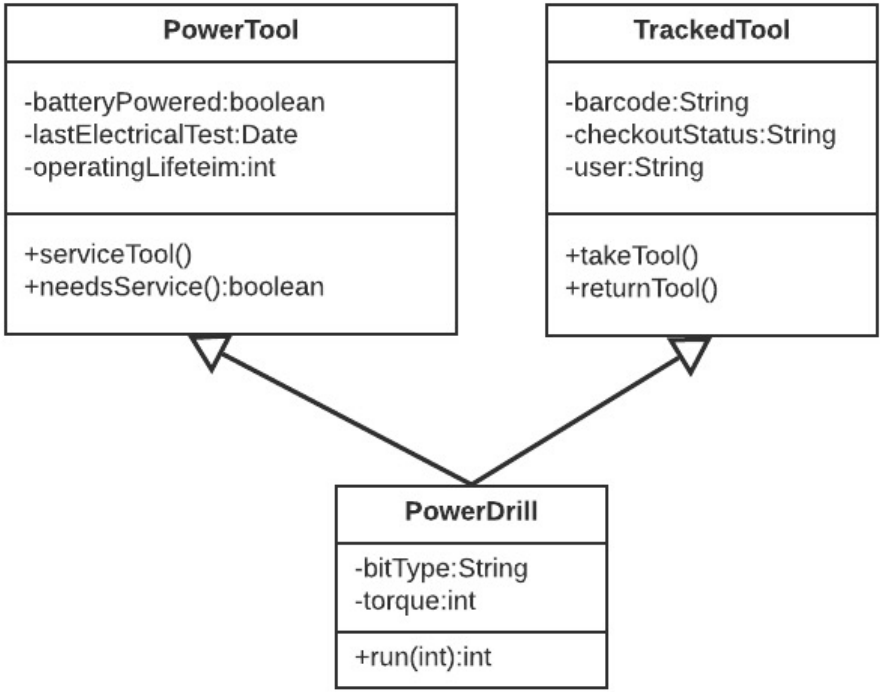
Multi-Level Inheritance



Hierarchical Inheritance

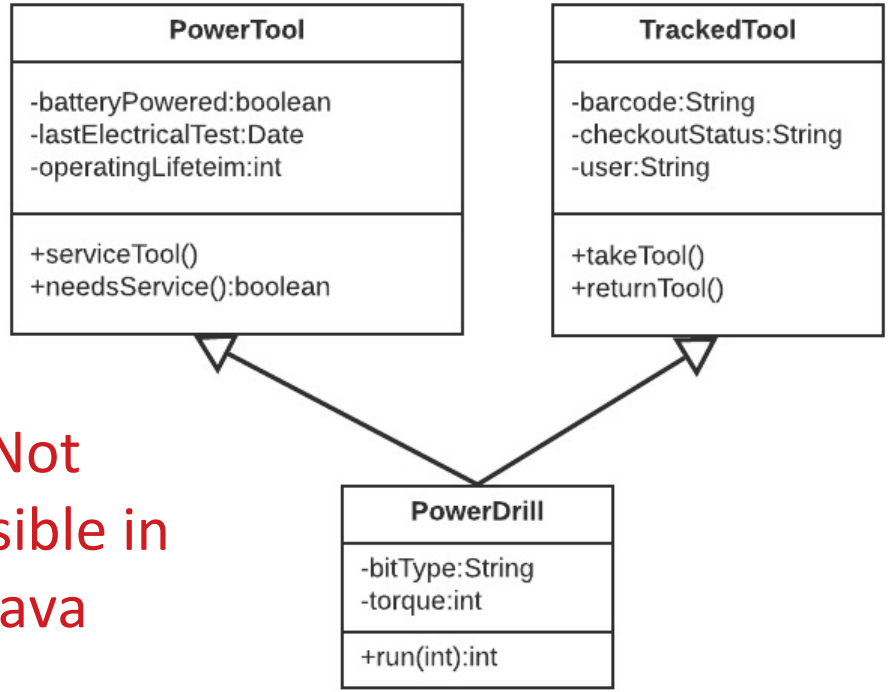


Multiple Inheritance



Multiple Inheritance

Not possible in Java



Abstract keyword

You hopefully noticed that the base PowerTool in the previous class was not an Abstract class

- The same is true of CardGame in the lab sessions which BlackJack extends

Abstract classes are ones that can not be instantiated, they let you design a super class which should not ever be able to exist without more information

- Abstract classes can still hold concrete implementations of some of their methods

Abstract methods in Java are similar but work for methods, they make a namespace for the method that can't be called without causing an error until a sub-class implements the method

University of Glasgow

Abstract classes in Java

```
public abstract class PowerTool
{
   // same code as before
}
```

Trying this will get an error because we cannot instantiate abstract classes:

```
PowerTool pt = new PowerTool();
```

Instead we must instantiate the class like this:

```
PowerTool pt = new Drill();
```

University of Glasgow

Abstract classes in Java

More commonly we see the abstract class used like this to allow us to work on lists of a similar type of item:

```
ArrayList<PowerTool> companyTools = new
                                ArrayList<PowerTool>;
tools.add(new Drill());
tools.add(new CircularSaw());
tools.add(new ImpactDriver());
```

We can do lot's of useful things with this list, like loops through it and find all the power tools that need to be serviced by calling PowerTools needsService()

University of Glasgow

Principle 4: Polymorphism

The ability for the same function name to do different things depending on the context it is used in.

- Polymorphism is strongly linked in to the concept of inheritance as the most common time to need to use it is to *override* the method implemented in a superclass
- Polymorphism is the most specific of the OO principles
- It's also the most critiques as well, not all developers think that it is important/necessary/useful to reuse the same names a lot

University of Glasgow

Reminder – Power Tool

```
public class PowerTool
{
  private boolean batteryPowered;
  private Date lastElectricalTest;
  private int operatingLifetime;

  // check safe to use, renew lastElectricalTest
  private void serviceTool()

  // check if power tool needs to be serviced
  private boolean needsService()

  // check if the tool breaks in use, return breakage details
  private String breakageCheck()
```

Polymorphism in Java - Overriding

PowerTool implements a needsService() method but perhaps CircularSaws are particularly dangerous and need to be serviced after every time they are used

```
public class CircularSaw extends PowerTool{

  boolean usedSinceService;

  // in CircularSaw, just returns usedSinceService
  private boolean needsService()

  // check safe to use, reset usedSinceService boolean
  private void serviceTool()
```

University of Glasgow

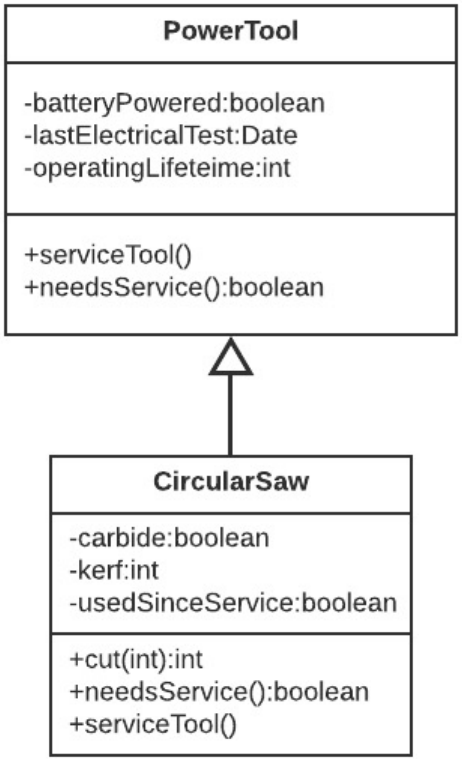
Polymorphism in Java - Overriding

This code will work just fine because java allows us to *override* a method

- Overriding is when a subclass replaces a method with it's own code
- Method calls of the class will use the new behaviour by default

You can still access the super classes method with the super keyword, for example, calling *super.needsService()* in the CircularSaw class will get a result based on the date last serviced from tool

Overriding Methods



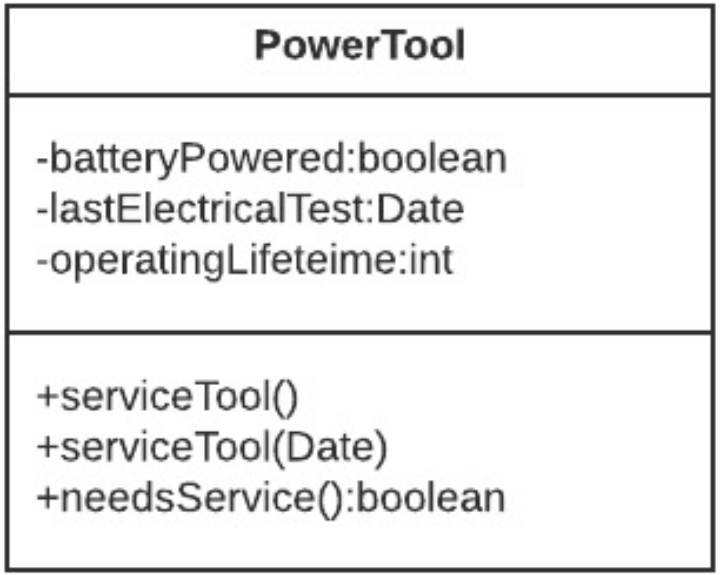
Polymorphism in Java - Overloading

A similar concept to overriding is that of overloading, this is when multiple methods in the same class share the same name but have different input parameters

```
public class PowerTool
{
   …
   // check safe to use, set lastElectricalTest to
   // todays date
   public void serviceTool()

   // sets lastElectrical to be dateOfService
   public void serviceTool(Date dateOfService)
```

Overloading Methods



University of Glasgow

Inheritance or Interfaces?

It probably hasn't escaped your notice that Abstraction and Inheritance seem quite similar

Java naming can add to this confusion, it is a bit unhelpful when learning OOP concepts because:

- Full Abstraction is realised by using the `interface/implements` keywords
- Inheritance is realised with the `abstract/extends` keywords but this only realises partial Abstraction!

With all this fuss, you might be wondering why we bother to use one or the other

University of Glasgow

Inheritance or Interface in Java?

Inheritance

1. Allows for code-reuse as subclasses get the code of the superclass
2. Can have abstracted elements with the abstract keyword
3. But can only allow inheritance from 1 class

Interfaces

1. Create blueprints to guide program development with no details of implementation needed at all
2. Can allow multiple blueprints to be implemented in a single class
3. But does not share any code

University of Glasgow

Why we use Abstraction

Imagine you are building a video game and you need some game world objects to:

- Know if they collide with other objects (collidable)
- Be affected by physics (physical)
- Be aware of players around them and react to them (awareable)
- Pathfind and move through the environment (pathable)
- Take damage (hitable)
- Deal damage (fightable)

University of Glasgow

Why we use Abstraction

```
public class Rock implements collidable

public class Cart implements collidable, physical,
                                                hitable

public class Child implements awareable, pathable

public class Pedestrian implements awareable,
                              pathable, hitable, fightable

public class Guard implements collidable, physical,
                     awareable, pathable, hitable, fightable
```

University of Glasgow

Outline

1. Inheritance – allows a subclass to reuse code from a superclass – good code-reuse but tied into a 1-1 relaitonship

2. Polymorphism – allows us to use code with the same name in the same place – the least significant of the principles and simplest

3. Abstraction or Inheritance – beware abstract keywords and full abstraction, understand why we like to use abstraction even though it doesn't get us to the point of code re-use