More about inheritance

Exploring polymorphism
Main concepts to be covered

• method polymorphism
• static and dynamic type
• overriding
• dynamic method lookup
• protected access
The inheritance hierarchy
Conflicting output

What we want

CD: A Swingin' Affair (64 mins)*
  Frank Sinatra
  tracks: 16
  my favourite Sinatra album

video: O Brother, Where Art Thou? (106 mins)
  Joel & Ethan Coen
  The Coen brothers’ best movie!

What we now have

title: A Swingin' Affair (64 mins)*
  my favourite Sinatra album

  title: O Brother, Where Art Thou? (106 mins)
  The Coen brothers’ best movie!
The problem

• The `print` method in `Item` only prints the common fields.

• Inheritance is a one-way street:
  - A subclass inherits the superclass fields.
  - The superclass knows nothing about its subclass’s fields.
Attempting to solve the problem
Static type and dynamic type

• A more complex type hierarchy requires further concepts to describe it.

• Some new terminology:
  - static type
  - dynamic type
  - method dispatch/lookup
Static and dynamic type

**What is the type of c1?**
Car c1 = new Car();

**What is the type of v1?**
Vehicle v1 = new Car();
Static and dynamic type

- The declared type of a variable is its static type.
- The type of the object a variable refers to is its dynamic type.
- The compiler’s job is to check for static-type violations, such as:

  ```java
  Item item = (Item) iter.next();
  item.print(); // Compile-time error.
  ```

  Code from database - looks for print method in item (item’s static type is item – and there is no print there)
Overriding: the solution

- Overriding: the solution
- print method in both super- and subclasses.
- Satisfies both static and dynamic type checking.
Overriding

- Superclass and subclass define methods with the same signature.
- Each has access to the fields of its class.
- Superclass satisfies static type check.
- Subclass method is called at runtime – it overrides the superclass version.
- What becomes of the superclass version?
Method lookup

Neither inheritance nor polymorphism. The obvious method is selected.
Method lookup

Inheritance but no overriding. The inheritance hierarchy is ascended, searching for a match.
Method lookup

```java
Item v1;

v1.print();
```

Polymorphism and overriding. The ‘first’ version found is used.
Method lookup summary

- The variable is accessed.
- The object stored in the variable is found.
- The class of the object is found.
- The class is searched for a method match.
- If no match is found, the superclass is searched.
- This is repeated until a match is found, or the class hierarchy is exhausted.
- Overriding methods take precedence.
Super call in methods

• Overridden methods are hidden ...
• ... but we often still want to be able to call them.
• An overridden method can be called from the method that overrides it.
  – super.method(...)
  - Compare with the use of super in constructors.
Calling an overridden method

```java
public class CD {
    ...
    public void print()
    {
        super.print();
        System.out.println("    " + artist);
        System.out.println("    tracks: " +
                           numberOfTracks);
    }
    ...
}
```
Method polymorphism

• We have been discussing *polymorphic method dispatch*.
• A polymorphic variable can store objects of varying types.
• Method calls are polymorphic.
  - The actual method called depends on the dynamic object type.
The Object class’s methods

• **Methods in Object are inherited by all classes.**

• **Any of these may be overridden.**

• **The `toString` method is commonly overridden:**

  - `public String toString()`

  - Returns a string representation of the object.
Overriding toString

```java
public class Item
{
    ...

    public String toString()
    {
        String line1 = title + " (" + playingTime + " mins")");
        if(gotIt) { return line1 + "*\n" + " " + comment + "\n";
        } else { return line1 + "\n" + " " + comment + "\n";
        }
    }
}
```
Overriding toString

- Explicit print methods can often be omitted from a class:
  - `System.out.println(item.toString());`

- Calls to println with just an object automatically result in toString being called:
  - `System.out.println(item);`
Protected access

- Private access in the superclass may be too restrictive for a subclass.
- The closer inheritance relationship is supported by *protected access*.
- Protected access is more restricted than public access.
- We still recommend keeping fields private.
  - Define protected accessors and mutators.
Access levels
Review

• The declared type of a variable is its static type.
  - Compilers check static types.
• The type of an object is its dynamic type.
  - Dynamic types are used at runtime.
• Methods may be overridden in a subclass.
• Method lookup starts with the dynamic type.
• Protected access supports inheritance.