Workshop 5

Table of Contents

Inheritance

- What is inheritance?
  - the ability of a child class to inherit attributes/methods of parent class
- What advantages does it give us as programmers?
  - useful abstraction, represent generalisation of similar objects, implementing only particulars in child class while sharing common attributes/methods
  - minimise code repetition, maximise code reuse
  - improve code maintainability
- What relationship does inheritance represent?
  - “is a”
- What is the super keyword? Where do we typically use it?
  - `super` refers to the parent class
  - typically used to invoke a method of the parent class, e.g. to invoke the parent constructor
- What is method overriding?
  - method overriding is creating a method in a child class with the same *signature* as the method in the parent class, such that you “override” the behaviour to meet the needs of the child
- What class does every class inherit from?
  - `Object`
- What are some methods inherited from this class, and why do we generally replace them?
  - `equals()`: define a meaningful equality condition, default is return false
  - `toString()`: make a meaningful string representation for our object (default prints class name and reference)

Abstract Classes

1. If you label a class or method as abstract, what does it do?

- class cannot be instantiated
- indicates implementation is not complete

2. What is the conceptual meaning of abstract classes?

- useful generalisation that is not attached to a real-world entity

3. How can we decide whether a class should be abstract or concrete?

- does the class represent a real-world entity?
- do the methods of the class make meaningful actions, or are they only being defined as a place-holder to be properly implemented by child class?
- is the logic of the class incomplete?

Polymorphism

1. Define polymorphism.

    - literally “many forms”
    - ability to use objects/methods in many ways

2. In what ways does Java allow polymorphism?

    - overloading: same method with various forms depending on *signature*
    - overriding: same method with various forms depending on *class*
    - substitution: using subclasses in place of superclasses

3. What is upcasting, and why is it useful to be able to write code like: `Piece[] pieces = new Piece[]{new Rook(), new King(), new Queen()}`

    - *upcasting* is the process of assigning a reference to a subclass to a variable of *parent-class* type
    - this allows you to refer to a generic parent class, without needing to know which child class it is in advance, making code much more general

4. What is downcasting? What do you need to be aware of when using it?

    - *downcasting* is casting a reference from a parent class to a child class
    - this will only work if the original object is actually of child class type