# cisc3120 design and implementation of software applications I spring 2015 lecture # 1.5

## topics:

- introduction to java, part 5
  - $\mbox{ code documentation with } javadoc$
  - inheritance
  - overriding methods
  - polymorphism
  - 'super' keyword
  - 'final' keyword

cisc3120-spring15-ozgelen-lecl.5

code documentation with javadoc - comments and common tags

- $\bullet$  descriptions have to be placed within /\*\*  $\ldots$  \*/ immediately before the class, method, field, etc.
- javadoc recognizes certain keywords that follows the @ character, referred to as tags
- some common tags:
  - @author [name]
  - Cparam [param\_name] [parameter description]
  - @return [description of returned data]
  - @throws [exception name] [exception description]
- to view the documentation, look for the index.html file within the documentation path if specified by -d option

# code documentation with javadoc • javadoc tool uses the special comments enclosed within /\*\* ... \*/ and generates html documenation, in the same format as Java API docs • regular comments in the code with // and /\* ... \*/ are ignored • javadoc can be used to describe classes, constructors, methods, fields and interfaces • usage: \$> javadoc [options] [files] • some useful options: -d <path>: saves the doc files in path -author : compiles the docs with the author information (if provided) -private : includes private fields and methods (default is public and protected) -classpath <path>: searches for references in path • example: \$> javadoc -d docs \*.java generates documentation for all .java files in the current directory and saves them in docs directory

cisc3120-spring15-ozgelen-lecl.5

## inheritance in java (1)

- classes by themselves are only serve to organize code. the real power of OOP lies in how these classes are related to one another through inheritance.
- *inheritance* is the means by which classes are created out of existing classes
- the idea is to re-use existing code and create *specialized* versions of existing classes when necessary
- in java a class can inherit from another class using the keyword extends, example:

```
class Animal {
  public float weight;
  public void eat() {...}
}
class Mammal extends Animal {
  public void breathe() {...}
}
```

cisc3120-spring15-ozgelen-lecl.5

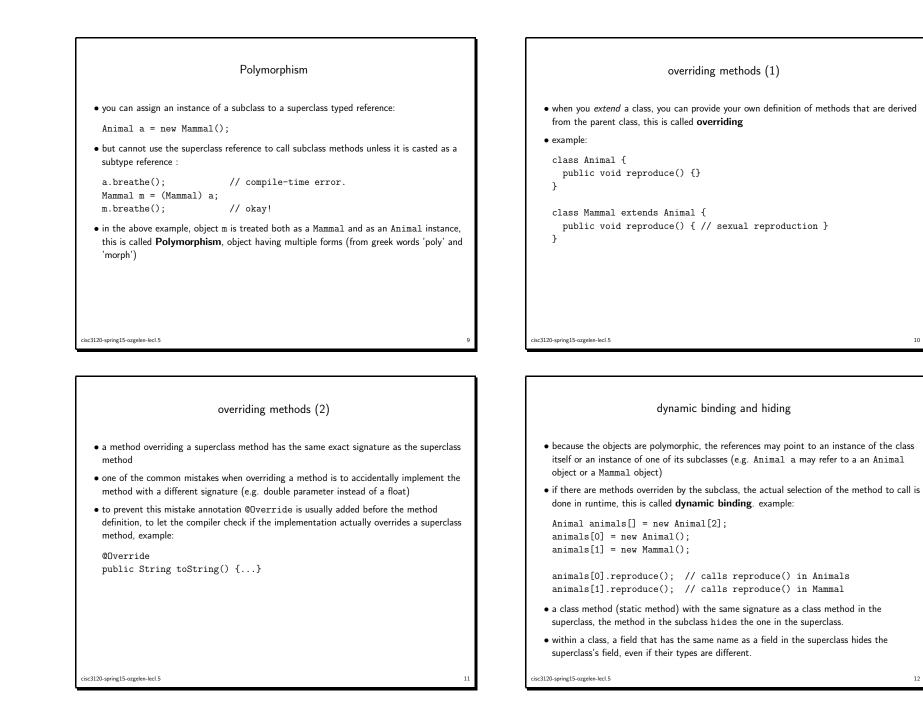
cisc3120-spring15-ozgelen-lecl.5

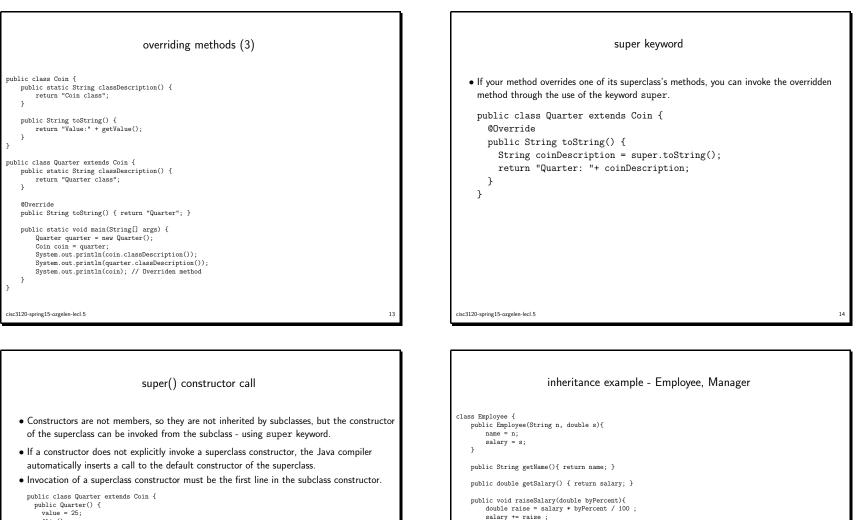
## inheritance in java (3) inheritance in java (2) • for the previous example Animal is said to be the *superclass* of Mammal and Mammal is • think of the most primitive Java class, Object as being at the root of the inheritance tree said to be the *subclass* of Animal • all other classes are "children" or *subclasses* of that class • class Mammal inherits the field weight and method eat() therefore the following • here is an example of the inheritance tree for Integer: statements would be legal: java.lang.Object Mammal human = new Mammal(); L human.weight = 80.0; // inherited from Animal +--java.lang.Number // inherited from Animal human.eat(); human.breathe(); // specific to Mammal +--java.lang.Integer • prefixes super and sub comes from set theory, to represent the relationship between the • Integer is a subclass of Number and Number is a subclass of Object instances e.g. all Mammal instances are Animals • Integer is also a subclass of Object • this is known as the *is-a* relationship between a subclass and the superclass • conversely a parent is also called a *superclass* • superclasses are also called *base* class or *parent* class • Object is a superclass of Number and Number is a superclass of Integer • subclasses are also called *derived* class or *child* class • Object is also a superclass of Integer cisc3120-spring15-ozgelen-lecl.5 cisc3120-spring15-ozgelen-lecl.5

| inheritance in java (4)  |
|--|
| • as you move DOWN the inheritance tree from the root to the leaf, you are <i>extending</i> subclasses from parent classes                               |
| • as you move UP the inheritance tree from the leaf to the root, you can say that each subclass instance is-a <i>more specific</i> version of its parent |
| • A subclass <b>does not</b> have access to the <b>private</b> members of its parent class:  |
| <pre>class Animal {    private float weight;    public void setWeight(float w) { weight = w; } }</pre>   |
| <pre>class Mammal extends Animal {    public gainWeight() {       weight += 10; // error!       setWeight(50); // okay!    } }</pre>                     |
| risc3120-spring15-ozgelen-lecl.5   |

- If the superclass has public or protected methods for accessing its private fields, these can also be used by the subclass.
- The protected modifier specifies that the member can only be accessed:
  - within its own package
  - $-\ensuremath{\,\text{by}}$  a subclass of its class in another package

cisc3120-spring15-ozgelen-lecl.5





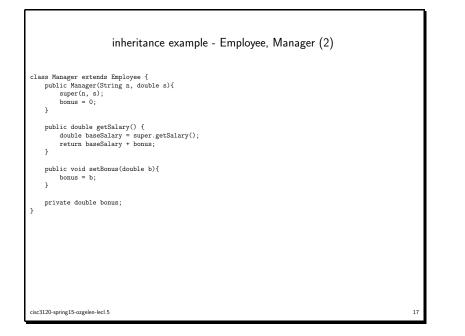
```
flip();
 }
OR
 public Quarter() {
   super(25);
 }
}
```

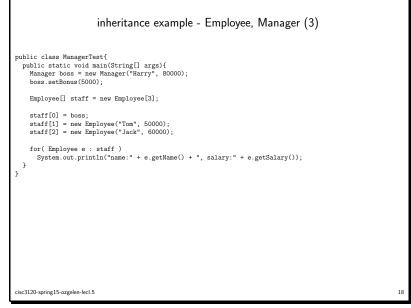
cisc3120-spring15-ozgelen-lecl.5

```
private String name;
private double salary;
```

cisc3120-spring15-ozgelen-lecl.5

ı





final classes and methods

. you can declare some or all of a class's methods final.
. you use the final keyword in a method declaration to indicate that the method cannot
be overridden by subclasses.
public class Coin {
 final int getDefaultFace() {
 return Coin.HEADS;
 }
 if you declare an entire class final this will prevent the class from being subclassed

cx202-profected.