CSN-103: Fundamentals of Object Oriented Programming

Instructor: Dr. Rahul Thakur

Assistant Professor, Computer Science and Engineering, IIT Roorkee

Method Overriding

- Method in a subclass has the same name and type signature as a method in its superclass
  - Version of the method defined by the superclass will be hidden
  - Version of that method defined by the subclass will be called

Subclass is said to ***override*** the method in the superclass

super and Method Overriding

- To access the superclass version of an overridden method
  - **super** Keyword: For immediate superclass
- Overloading vs. Overriding
  - Overriding occurs *only* when the names and the type signatures of the two methods are identical
  - Otherwise, the two methods are simply overloaded

Dynamic Method Dispatch

- The mechanism by which a call to an overridden method is resolved at the **runtime**
  - That's how Java implement **runtime polymorphism**
- Recap: Superclass reference variable can refer to a subclass object
  - Used to resolve calls to overridden methods at the **runtime**
  - Java determine which version of method to execute based on the **type of object referred at the time of call**

Why Overridden Methods?

- Allow a general class to specify methods that will be common to all its derivatives (subclasses)
  - Subclasses can define **their own implementation** for some of these methods
  - One Interface, Multiple Methods : Polymorphism
- Also defines those methods that a derived class must implement by its own
  - Abstract Class
  - Enforces a consistent interface

Why Overridden Methods?

- Dynamic, run-time polymorphism is one of the most powerful mechanism of OOP
- Provides you the ability to call methods on instances of different classes
  - Without recompiling the code
  - While maintaining a clean abstract interface

Abstract Class and Method

- Situations where you wish to define (just) the structure of a class
  - Without providing the implementation of every method
  - Leaving the implementation of methods to subclasses
- Situations where superclass can't create a meaning implementation
  - Example: **area()** method in the Figure Class

Abstract Method

- How to handle these situations
  - Just print a warning message in the method of base class
  - Ensure that subclass does override the method → Abstract Method
- OR Use **abstract** modifier to force the implementation of a method by the subclasses

Abstract Class

- Abstract method has **no implementation** specified in the superclass
- General form of abstract method

```
abstract type name(parameter-list);
```

- Abstract class: A class that has one or more abstract methods

Abstract Class

- A class is declared abstract by simply using the **abstract** keyword in front of class keyword
- There can be no object of abstract class
  - Can't use **new** operator to instantiate an object
  - Object would be useless because abstract class is not fully defined
- However, we can create object references
  - These references can be used to point to subclass object → Achieve runtime polymorphism
- Example: abstract class A

```
A obj = new A();          // Error
A obj;                    // Valid
```

AbstractDemo.java AbstractAreas.java

Abstract Class

- Is it necessary that derived class must implement the abstract methods of the base class??
  - Yes, **unless** the derived class itself declared abstract
- You can not have abstract constructors or abstract static methods

Final Keyword

- Final has three uses
  - **Final variables**
  - To **disallow** a method to be overridden by the subclass
  - To **prevent** a class from being inherited

$\left.\begin{array}{l} \\ \\ \end{array}\right\}$ **Inheritance**

```
class A {
      final void meth() {
                System.out.println("This is a final method.");
      }
}

class B extends A {
      void meth() {                                // ERROR! Can't override.
                 System.out.println("Illegal!");
      }
}
```

Final to Prevent Inheritance

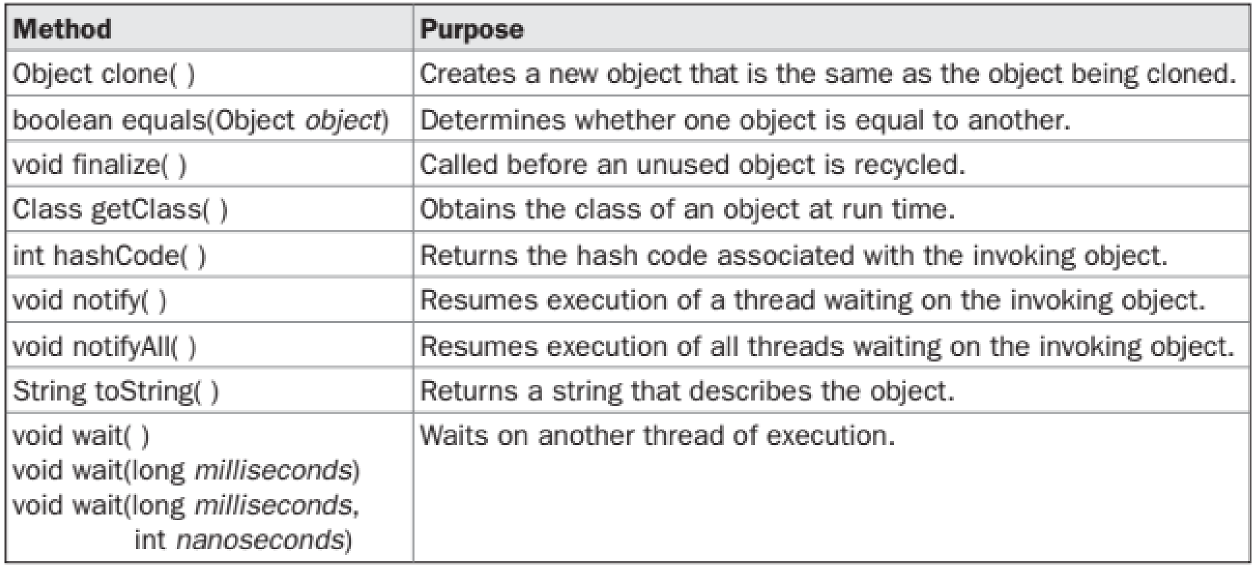
```
final class A {
      // ...
}

class B extends A {            // ERROR! Can't subclass A
      // ...
}
```

- Declaring a class as final **implicitly** declare **all** its methods as final, too

Object Class

- Special Java class: Object
- All other classes are subclasses of Object class
- Reference variable of Object class can refer to object of any other class (even arrays)

Object Class



Method	Purpose
Object clone( )	Creates a new object that is the same as the object being cloned.
boolean equals(Object *object*)	Determines whether one object is equal to another.
void finalize( )	Called before an unused object is recycled.
Class getClass( )	Obtains the class of an object at run time.
int hashCode( )	Returns the hash code associated with the invoking object.
void notify( )	Resumes execution of a thread waiting on the invoking object.
void notifyAll( )	Resumes execution of all threads waiting on the invoking object.
String toString( )	Returns a string that describes the object.
void wait( ) void wait(long *milliseconds*) void wait(long *milliseconds*, int *nanoseconds*)	Waits on another thread of execution.

- *getClass(), notify(), notifyAll(),* and *wait()* are declared as final
- Other methods can be overridden