Abstracts

New Austrian Coding School

Programming - memo

An abstract is a **template** for a general idea that is **not yet finished**. Because it is **unfinished**, we **cannot** create an object from it. A **class** that uses that template has to at least **define** the unfinished things. This class can have on its own **more** things than just the template if it needs to.

Interfaces and abstracts

Interfaces:	Abstracts:
○ Cannot have attributes	○ Can have attributes
○ Cannot implement methods	○ Can implement methods
○ All methods are public	○ You choose the method visibility
○ Need another class to be used	○ Need another class to be used
○ Promotes class independence	○ Promotes class dependence

How to define an abstract

- We write **public abstract class** followed by the name
- **Everything** else is exactly like normal classes
- Abstract means **unfinished**, and therefore a **new** Transport object **cannot** be created
- This abstract class represents a **template** for transports

```
public abstract class Transport {

  private String engine;

  public Transport(String engine) {
     this.engine = engine;
  }

  public void drive(){
     System.out.println("Driving");
  }
}
```

How to extend from an abstract

- We create a class as usual
- We add **extends** plus the name of the abstract
- We add the **constructor** if the abstract had one
- We can use everything from Transport that was **not private**
- We can **add** anything else we want, like the **fly** method

```
public class Plane extends Transport {

  public Plane(String engine) {
     super(engine);
  }

  public void fly(){
     System.out.println("Flying");
  }
}
```

Abstract methods

An abstract method is something that the template **does**, but it's still **uncertain** how it should do it. They define the *what* but not the *how*. Basically they are the same as **interface methods**, but you can choose the **visibility**. Every class that **extends** from this abstract must **define** its abstract methods.

Abstract methods

○ We write the visibility
○ We write the keyword **abstract**
○ We write the return type
○ We write the name
○ We write the arguments
○ We end with a **semicolon**
○ Almost the same as interface methods
	`public abstract class Kid {`
	
	`  public abstract void annoy();`
	
	`}`

○ We define all **abstract** methods of the class from which we **extend**
○ Defined exactly the same as with **interface methods**
	`public class Tommy extends Kid {`
	`  @Override`
	`  public void annoy() {`
	`     System.out.println("Are we there yet?");`
	`  }`
	`}`

The protected visibility

Review of the **four** types of visibility **limitations**:

- **public** - available to every class
- **default** - available only to classes organized in the same package
- **private** - available only to the same class
- **protected** - available to classes organized in the same package and to classes that extend from that class, regardless of the package where they are