OOP with Java

Yuanbin Wu
cs@ecnu

OOP with Java

- 通知
  - Project 4: 4 月 19 日晚 9 点

- 复习
  - 类的复用
  - 组合 (composition):
    - has-a 关系

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MyCompType {
    private MyType m = new MyType();
    private String s;
    public MyCompType(){
        s = new String("Hello");
    }
}
```

- 复习
  - 继承 (inheritance)
    - is-a 关系

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{

    String s = new String("Hello");
    public double add(double d){return this.d + d;}
    public double add(String s){return this.s + s;}

    public void set(double x){ i = (int)x; }
    public double get() { return i; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        System.out.println(ms.get());
        System.out.println(ms.add(1.0));
        System.out.println(ms.add("World"));
    }
}
```

- 复习
  - 继承
    - 子类有父类的所有方法和数据
    - 子类可以定义新的方法和数据
    - 子类可以重写 (override) 父类的方法
  - super 关键字
  - 每一个子类对象都隐含包含一个父类对象
  - Object 对象
    - Single root class hierarchy tree
    - 方法 :

```
boolean equals(Object o)

String toString()
```

```
class MyType {

    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public double get() { return d; }

    public static void main(String [ ]args){
        MyType m = new MyType();
        MyType n = new MyType();
        String s = “hello”;
        m.equals(n);
        m.equals(s);
    }
}
```

OOP with Java

- protected
- upcasting
- final 关键字

protected

- 访问控制
  - package access
  - public
  - private

protected

- 函数重写

```
class MyType {
    public int i;
    public double d;
    public char c;
    public void set(double x) { d = x;}
    public void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{

    public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

protected

- 函数重写？

```
class MyType {
    public int i;
    public double d;
    public char c;
    private void set(double x) { d = x;}
    private void set(int y) {i = y;}
    private double get() { return d; }
}
```

```
public class MySubType extends MyType{

    // can not access!!
    // public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

Protected

- 父类的方法
  - public
  - private
  - 是否有可能被子类访问而不被外界访问？

protected

- protected
  - 可以被子类 / 同一包中的类访问，不能被其他类访问
    - 弱化的 private
    - 同时赋予 package access

protected

```
class MyType {
    public int i;
    public double d;
    public char c;
    protected void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    protected double get() { return d; }
}
```

```
public class MySubType extends MyType{

    public double foo(){ return get(); }
    public void set(double x){ i = (int)x; }
    public void set(char z) {c = z; }

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        ms.set(1.0);
        System.out.println(ms.get());
        System.out.println(ms.i);
        System.out.println(ms.d);
    }
}
```

Protected

- 访问控制
  - package access
  - public
  - private
  - protected

Upcasting

- 继承
  - is-a 关系
  - 子类有父类所有的数据和方法
  - 类型关系：子类是一种父类
    - the sub-class is a type of the base class

Upcasting

- 例子

```
class Instrument {
    public void play() {}
    static void tune(Instrument i) {
        // ...
        i.play();
    }
}

public class Wind extends Instrument {
    public static void main(String[] args) {
    Wind flute = new Wind();
    Instrument.tune(flute);
    }
}
```

Upcasting

Upcasting

- 例子

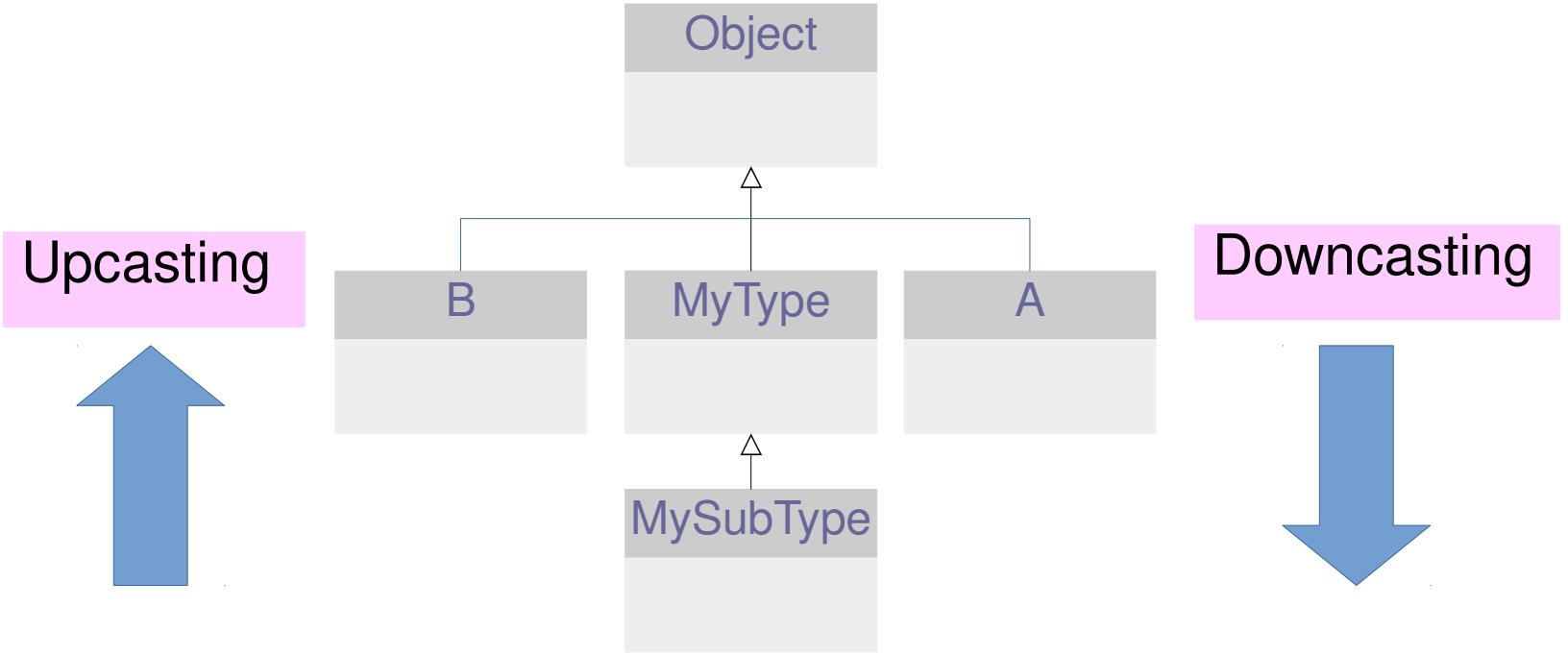
```
public class MySubType extends MyType{

    String s = new String(“Hello”);
    public double add(double d){return this.d + d;}
    public double add(String s){return this.s + s;}

    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;

        System.out.println(m.get());
        System.out.println(ms.add("World"));

        m.set(1.0);
        System.out.println(m.get());
        System.out.println(ms.get());
    }
}
```

Upcasting

- Upcasting
  - 需要父类对象
    - 引用，函数参数
  - 可以用子类对象带入
  - 安全的类型转换
    - 子类拥有父类所有的数据和方法

Upcasting

- Upcasting



Upcasting

- 子类重写了父类方法？

```
class MyType {
    public int i;
    public double d;
    public char c;
    protected void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

多态

```
public class MySubType extends MyType{
    public void set(double d){
        System.out.println("Sub-class set");
        i = int(d);
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}
```

Upcasting

- 类型转化
  - 基本类型
    - int → double (安全 , 自动转换)
    - double → int (损失精度 , 强制转换)
  - 基本类型与 wrapper
    - int → Integer (autoboxing)
    - Integer → int (unboxing)
  - 类
    - 不支持强制转化
    - 子类 → 父类 (安全 , upcasting)
    - 父类 → 子类 (downcasting)

Upcasting

- Downcasting
  - MySubType ms = (MySubType)m;
  - 仅在 m 确实指向子类对象时才能进行
  - 运行时类型信息 (RTTI)
- 例子 :
  - 重写 equals 方法

Upcasting

- 总结
  - 子类是一种父类 (is-a)
  - 父类的引用可以指向子类对象

final 关键字

- final 关键字
  - 不同的环境下有不同含义
  - 基本意义为：**不能被改变**

final 关键字

- final 数据
  - 编译时常数
  - 一旦被赋值就不能被修改

final 关键字

- final 数据
  - 例子

```
class MyType {
    public int i;
    public final double d = 1;
    public char c;
    public double get() { return d; }
    public void set(double x) {d = x;}

    public static void main(String []args){
        MyType m = new MyType();
        // m.d = 2.0;
    }
}
```

final 关键字

- final 数据
  - final 引用

```
class MyType {
    public int i;
    public final double d = 1;
    public char c;
    public final int [ ] a = new int[10];

    public double get() { return d; }
    public void set(double x) {d = x;}
    public static void main(String []args){
        MyType m = new MyType();
        m.a[0] = 1.0;
        //m.a = new int[10];
    }
}
```

final 关键字

- final 数据
  - final + static
    - static final int i = 1;
  - 仅有一个不可变的存储空间

final 关键字

- final 数据
  - Blank final

final 成员在定义时可以不给初值
必须在构造函数中初始化

```
class MyType {
    public int i;
    public final double d;
    public char c;
    public double get() { return d; }
    public MyType(double x){ d = x;}

    public static void main(String []args){
        MyType m = new MyType(1.0);
        System.out.println(m.get());
        // m.d = 2.0;
    }
}
```

final 关键字

- final 参数
  - 函数不能修改参数的引用 .

```
class FinalArgs {
    public static void set(final int [ ] a) {
        a[0] = 1;
        // a = new int [10];
    }
    public static void main(String []args){
        int [ ]a = new int[10];
        FinalArgs.set(a);
    }
}
```

final 关键字

- final method
  - 不能被重写

```
class MyType {
    public int i;
    public double d;
    public char c;
    final void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
public class MySubType extends MyType{
    // can't overide
    /* public void set(double d){
        System.out.println("Sub-class set");
        i = int(d);
    } */
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}
```

final 关键字

- final class
  - 不能被继承

```
final class MyType {
    public int i;
    public double d;
    public char c;
    final void set(double x) { d = x;}
    protected void set(int y) {i = y;}
    public double get() { return d; }
}
```

```
// can not be extended
/*
public class MySubType extends MyType{
    public void set(double d){
        System.out.println("Sub-class set");
        i = int(d);
    }
    public static void main(String [ ]args){
        MySubType ms = new MySubType();
        MyType m = ms;
        m.set(1.0);
    }
}*/
```

final 关键字

- 不可变 (immutable)
  - 不可变对象：一旦创建就不能更改其状态
  - 优点：易于使用，易于 debug，易于维护
  - 缺点：空间 / 时间消耗
- final
  - 帮助构造不可变对象