Common items go to the super class and the individual ones are to be found inside the subclasses

```
class Chair {
    String color;
}

class Room {
    Chair[] chairs;
}

class ConferenceTable {}

class ClassRoom extends Room{

}

class MeetingRoom extends Room{
    ConferenceTable table;
}
```

Variable names should be meaningful

```
// bad example
int a = 10;
int b = 20;
int height = 9;
double area = (a + b) / 2 * h;
```

This is good

```
// GOOD
int arm1 = 10;
int arm2 = 20;
int height = 9;
```

```
double area = (arm1 + arm2) / 2 * height;
```

Condition inside loop/if should be made easier to read by replacing long chains of conditions with a method

```
// Bad code
class Student {
    double attendancePercentage;
    double CGPA;
    boolean isRegularStudent;
}


class HelloWorld {
    public static void main(String[] args) {

        if(student.attendancePercentage >= 80 && student.CGPA >=
3.80 && student.isRegularStudent) {
            student.assignTA();
        }

    }
}
```

```
// GOOD
class Student {
    double attendancePercentage;
    double CGPA;
    boolean isRegularStudent;

    public boolean isValidStudent() {
        return this.attendancePercentage >= 80 && this.CGPA >=
3.80 && this.isRegularStudent;
    }
}
```

```
class HelloWorld {
    public static void main(String[] args) {

        if(student.isValidStudent()) {
            student.assignTA();
        }

    }
}
```

Related items should be put together inside a new class (encapsulation)

```
// BAD
class Student {
    double attendancePercentage;
    double CGPA;
    boolean isRegularStudent;
    String course1; double course1GPA;
    String course2; double course2GPA;

    //           ....

    String course10; double course10GPA;
}
```

```
// GOOD
class Course {
    String courseName; double GPA;
}
```

```
class Student {
    double attendancePercentage;
    double CGPA;
    boolean isRegularStudent;
    Course courses[10];
}
```

Getters and setters should be used to prohibit unintended modifications

```
// GOOD
class Course {
    String courseName; private double GPA;
    public void setGPA(double gpa) {
        if(gpa > 4.00 || gpa < 0.00) return;
        this.GPA = gpa;
    }
}
```

Instead of using types to differentiate between different class types, use inheritance.

```
// BAD
class Student {
    int type;
    int FIRSTYEAR=1;
    int SECONDYEAR=2;
    int THIRDYEAR=3;
    int FORTHYEAR=4;

    public void attendProgram() {
        if(type == FIRSTYEAR) {
           behaveLikeFirstYear()
        } else if (type == SECONDYEAR) {
           behaveLikeSecondYear();
        } else {
            // ...
```

```
        }
    }

}
```

```
// GOOD
class Student {
behave();
}

class FirstYearStudent extends Student {
@Override
behave();
}
class SecondYearStudent extends Student {
@Override
behave();
}
class ThirdYearStudent extends Student {
@Override
behave();
}
class ForthYearStudent extends Student {
@Override
behave();
}
```

Local extension

```
class Clock {
    private String logo = "Mouse.png";
}
// Local extension
class UIUClock extends Clock{
```

```
    private String logo = "UIU.png";
    Public void behaveLikeADigitalClock() {

    }
}
```

Replace subclass with fields

```
// BAD
class Clock {
    private String logo = "Mouse.png";
}

class UIUClock extends Clock{

    private String logo = "UIU.png";
}

class DUClock extends Clock {
    private String logo = "du.png";
}
```

```
class Clock {
    private String logo;
    private String varsityName;
}
```

When to create subclasses and when not to

Jerry is different from Nibbles

```
class Mouse {
    public void eat()
}
class Jerry extends Mouse {
    @Override
    public void eat() {}
}
class Nibbles extends Mouse {
    @Override
    public void eat() {}
}
```

```
class Cat {
    private String hatColor;
    private String torsoColor;
    private String trousersColor;
    public void attack() {}

}
```

Push down

```
// BAD
class Animal {
    public void regurgitate();
}

class Cow extends Animal {

}

class Cat extends Animal{
}
```

```
// Good
class Animal {}

class Cow extends Animal {
    public void regurgitate();
}

class Cat extends Animal{
}
```

Pull up methods

```
// BAD
class Animal {

}

class Cow extends Animal {
    public void eat();
}

class Cat extends Animal{
    public void eat();
}
```

```
// GOOD
class Animal {
```

```
    public void eat();
}
class Cow extends Animal {}
class Cat extends Animal{}
```

Related parameters should be encapsulated in a new class.

```
// BAD
void carryItems(Pencil pencil, Pen pen, Sharpner sharpner) {

 }
```

```
// GOOD
class PencilBox {
    Pencil pencil;
    Pen pen;
    Sharpner sharpner;
}
void carryItems(PencilBox pencilbox) {

    }
```

Replace error codes with exception

```
// BAD
// return -1 when the temperature is too hot
    static void carryElectricity(Wire wire) {
        // some code
        if(wire.temp >= 250) {
            return -1;
        }
        // some other codes
        return 0;
    }
```

```
// GOOD
static void carryElectricity(Wire wire) {
        // some code
        if(wire.temp >= 250) {
            throw new OverTempException();
        }
        // some other codes
        return 0;
    }
```