Concordia University

Engineering and Computer Science

SOEN 6441 – Advance Programming Practices

Summer 2019

Build 2: Coding Conventions

Submitted to – Prof. Nagi Basha

Team Member	ID
Ankur Malhotra	40041056
Raghav Kaushal	40075268
Subhannita Sarcar	40059367
Kritika Kritika	40068982
Shriyans Athalye	40037637

Contents

1. Introduction:

Code standards set prescriptive rules for proper code writing in terms of indentation, comments, file organisation, naming and declaration. It benefits original developers, peer reviewers, testers and future maintainers. Apart from improving internal quality of software it helps improve readability and thus productivity of software. It aims to increase sustainability of software by reducing cost to add new code to any existing code base.

According to *Java Code Convention* given by Oracle, the BattleShip game adheres to every category of code standard that must be followed in order to have a good quality software.

2. File Naming:

All the java files are named in order to enhance readability experience for everyone accessing or working with them. The java source file use suffix xyz.java and the compiled bytecode takes xyz.class. The following table shows files of Battleship game that are named according to the main functionality they provide, thus adhering to the file naming convention standards.

Coordiante.java
Game.java
Player.java
Screen.java
Ship.java
AttackBoard.java
SelfBoard.java

Table 1: File naming convention

3. File Organization:

No file in BattleShip game has lines of code more than 2000 as lines exceeding this number makes files cumbersome. In addition to every section of file is separated by blank lines and optional comments so that they can be easily identified.

Following are the LOC of each file which were written for battleship game

Coordiante.java	47
Game.java	197
Player.java	289
Screen.java	214
Ship.java	84
AttackBoard.java	234
SelfBoard.java	435
Alignment.java	9
Computer.java	363
FleetAttack.java	93
ShipStatus.java	11
DataValue.java	11
Timer.java	126

Table 2: LOC count of project files

4. Comments:

The general order of a source file is:

beginning comments, package and import statements and finally class and interface declarations.

Following is the sample of one of the source files that adheres to proper file organisation.

Screen.java

```
        /*                                      //1)Beginning Comments

        * Screen*
        * Version info *
         * Copyright notice
        */                                      //2)import statements
```

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
/**
 * Class representing screen of each player
 * @author Group 3
 * @version 1.2
 */
public class Screen extends JFrame implements ActionListener
```

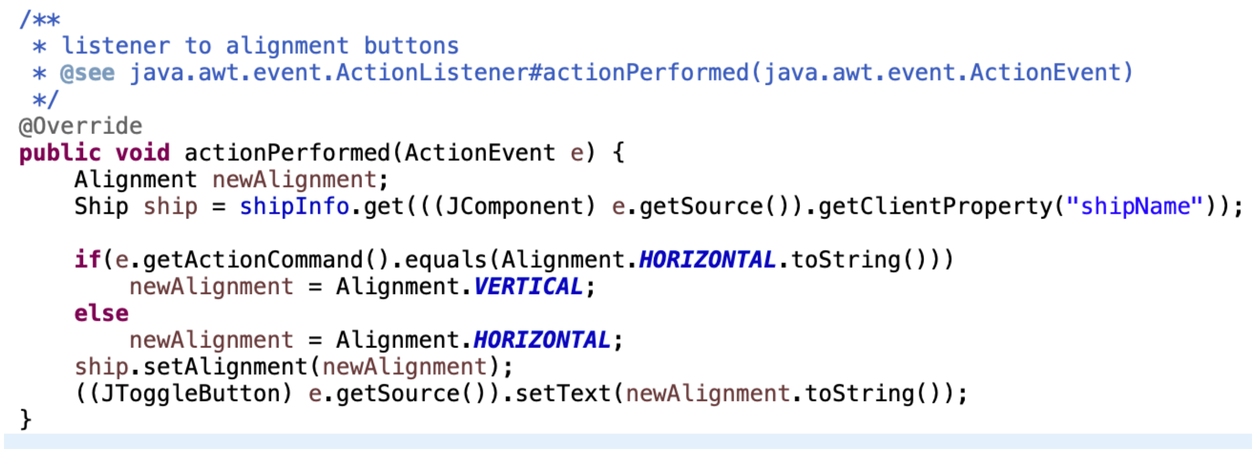
5. Indentation:

The standard states that lines must be properly indented. Exact construction of indentation (spaces vs tabs) is unspecified. But according to Oracle code convention, tabs must be set exactly every 8 spaces but not 4.

The line length is less than 80 characters as a thumb rule. When the expression doesn't fit in one line then line is broken according to the following rule [1]:

- Break after a comma
- Break after an operator
- Prefer high-level breaks to lower-level breaks
- Align the new line with beginning of the expression at the same level in the previous line
- If the above rules lead to confusing code or to code that's squished up against the right margin, just indent 8 spaces instead.

Following is a sample code from SelfBoard.java that breaks the method calls properly:

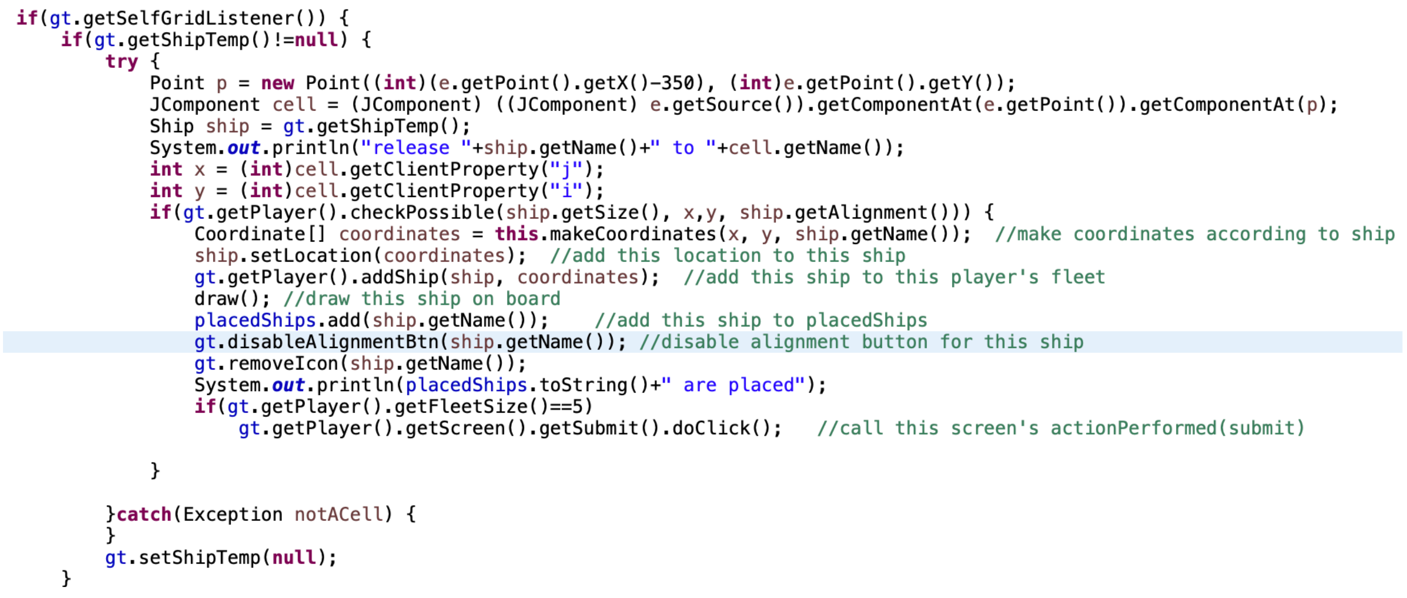


```
/**
 * listener to alignment buttons
 * @see java.awt.event.ActionListener#actionPerformed(java.awt.event.ActionEvent)
 */
@Override
public void actionPerformed(ActionEvent e) {
    Alignment newAlignment;
    Ship ship = shipInfo.get(((JComponent) e.getSource()).getClientProperty("shipName"));

    if(e.getActionCommand().equals(Alignment.HORIZONTAL.toString()))
        newAlignment = Alignment.VERTICAL;
    else
        newAlignment = Alignment.HORIZONTAL;
    ship.setAlignment(newAlignment);
    ((JToggleButton) e.getSource()).setText(newAlignment.toString());
}
```

6. Comments:

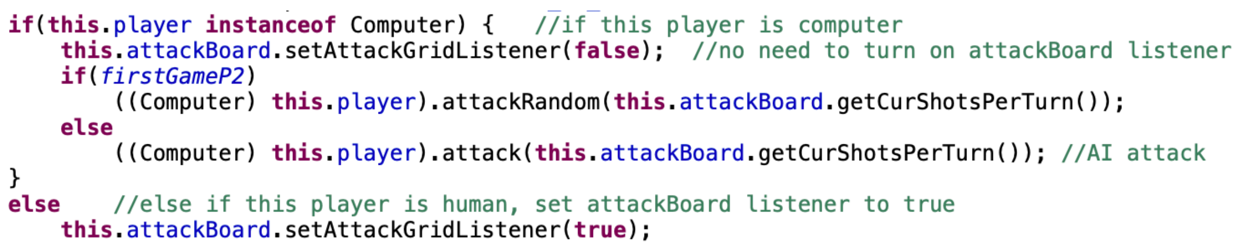
Every source file has implementation as well as documentation comments. The implementation comments are the // and /*..*/ that describe a particular implementation and documentation comments are javadocs. Javadoc can be converted to HTML files using javadoc tool. Javadoc describes code specification that is in a way free from implementation perspective. The aim of javadoc is to provide necessary information to new developers joining the team who do not necessarily have source code in hand. On the other hand, comments provide information that cannot be easily comprehended from the code.

Implementation comments in SelfBoard (Single Line Comment)



```
if(gt.getSelfGridListener()) {
    if(gt.getShipTemp()!=null) {
        try {
            Point p = new Point((int)(e.getPoint().getX()-350), (int)e.getPoint().getY());
            JComponent cell = (JComponent) ((JComponent) e.getSource()).getComponentAt(e.getPoint()).getComponentAt(p);
            Ship ship = gt.getShipTemp();
            System.out.println("release "+ship.getName()+" to "+cell.getName());
            int x = (int)cell.getClientProperty("j");
            int y = (int)cell.getClientProperty("i");
            if(gt.getPlayer().checkPossible(ship.getSize(), x,y, ship.getAlignment())) {
                Coordinate[] coordinates = this.makeCoordinates(x, y, ship.getName());  //make coordinates according to ship
                ship.setLocation(coordinates);  //add this location to this ship
                gt.getPlayer().addShip(ship, coordinates);  //add this ship to this player's fleet
                draw(); //draw this ship on board
                placedShips.add(ship.getName());    //add this ship to placedShips
                gt.disableAlignmentBtn(ship.getName()); //disable alignment button for this ship
                gt.removeIcon(ship.getName());
                System.out.println(placedShips.toString()+" are placed");
                if(gt.getPlayer().getFleetSize()==5)
                    gt.getPlayer().getScreen().getSubmit().doClick();   //call this screen's actionPerformed(submit)

            }

        }catch(Exception notACell) {
        }
        gt.setShipTemp(null);
    }
```

Fig 1: Implementation of comments in SelfBoard.java



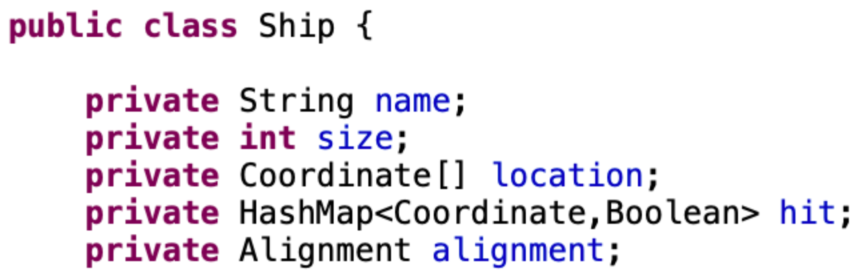
```java
if(this.player instanceof Computer) {   //if this player is computer
    this.attackBoard.setAttackGridListener(false);  //no need to turn on attackBoard listener
    if(firstGameP2)
        ((Computer) this.player).attackRandom(this.attackBoard.getCurShotsPerTurn());
    else
        ((Computer) this.player).attack(this.attackBoard.getCurShotsPerTurn()); //AI attack
}
else    //else if this player is human, set attackBoard listener to true
    this.attackBoard.setAttackGridListener(true);
```

Fig 2: Implementation of comments in Screen.java

7. Declaration:

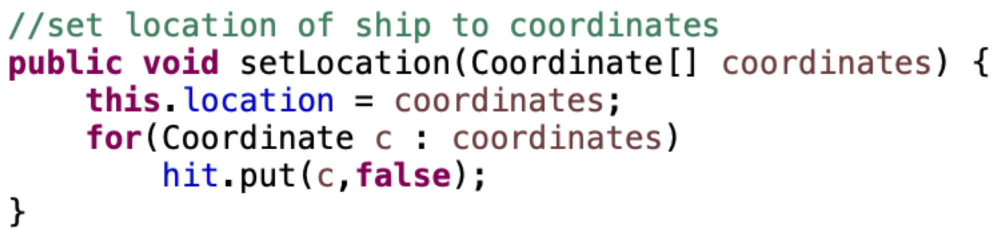
1) Number per line: In general, one declaration per line is recommended. Every source file of the game
   follows this rule.

Ship.java



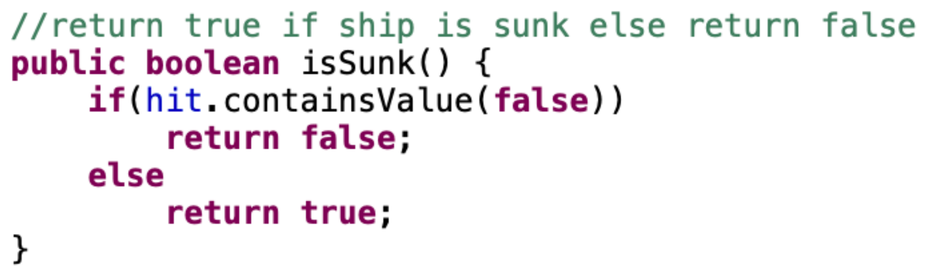
```java
public class Ship {

    private String name;
    private int size;
    private Coordinate[] location;
    private HashMap<Coordinate,Boolean> hit;
    private Alignment alignment;
```

2) Placement: The placement standard states that the declarations should be put in the beginning of the block except the for loop.



```java
//set location of ship to coordinates
public void setLocation(Coordinate[] coordinates) {
    this.location = coordinates;
    for(Coordinate c : coordinates)
        hit.put(c,false);
}
```

3) Class and Interface Declaration: According to Oracle: When coding Java classes and interfaces, the following formatting rules should be followed: • No space between a method name and the parenthesis “(“ starting its parameter list • Open brace “{” appears at the end of the same line as the declaration statement • Closing brace “}” starts a line by itself indented to match its corresponding opening statement, except when it is a null statement the “}” should appear immediately after the “{“ [1].



```
//return true if ship is sunk else return false
public boolean isSunk() {
    if(hit.containsValue(false))
        return false;
    else
        return true;
}
```

8.White Statements:

1) Blank Lines[1]: The Oracle convention says Blank lines improve readability by setting off sections of
code that are logically related. Two blank lines should always be used in the following circumstances:

• Between sections of a source file
• Between class and interface definitions One blank line should always be used in the following circumstances
• Between methods
• Between the local variables in a method and its first statement.

2) Blank Spaces [1]: A keyword followed by a parenthesis should be separated by a space. Example:
while (true){ ... } Note that a blank space should not be used between a method name and its
opening parenthesis. This helps to distinguish keywords from method calls.
• A blank space should appear after commas in argument lists.

Following source file Coordinate.java provides proper blank lines and white spaces.

```java
public class Coordinate {


        private int x;
        private int y;

        public Coordinate(int x, int y) {
                   this.x = x; this.y = y;
        }
        public int getX(){
            return x;
        }

        public int getY(){
             return y;
        }

//compare coordinate objects
        public boolean compareCoord(Coordinate coordinate){
              if(coordinate.getX() == this.x && coordinate.getY() == this.y){
                return true;
              }
           return false;
          }

    }
```

9.Naming Convention:

The names should be given to make programs more understandable and easier to read. They can also give information about the function of the identifier—for example, whether it's a constant, package, or class—which can clarify the code [1].

Components	Convention	Examples
Classes	Name should be noun; beginning with capital letters and every subsequent word with a capital letter.	Coordinate, Game, Player, Screen, Ship
Methods	should be verb with first letter as lowercase, subsequent word in uppercase.	compCoord, getX, getY, p1Play, p2Play, setTakeTurnAttack, getTakeTurnAttack, getScreen, isOvelap,
Variables	All instances, classes, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters.[1]	int x, int y, Player p1, Player p2, Screen screen, int[][] attackData etc. all follow this convention.
Constants	Constants should be declared in capital letters with underscore between names	Not used so far

Table 3: Table of Naming convention

References:

[1] Code Conventions for the Java Programming Language: Contents: Technical Report: https://www.oracle.com/technetwork/java/index.html

[2] Google Java Coding Style. [Available] Online: https://google.github.io/styleguide/javaguide.html

[3] Java Code Convention, Oracle Corp.2007. [Available]Online: https://www.oracle.com/technetwork/java/codeconventions-150003.pdf