CODING STANDARDS

JAVA Source File:

- The source file is organized with documentation comment, package declaration, followed by a class comment. Imports groups (static last), class/interface signature and so on as shown below.

```
package main.java;

import java.io.File;
import java.util.Scanner;

/**
 * Responsible for playing the game.
 * Covers tasks ranging from 'map editing' to 'actual gameplay'.
 * Responsible for only interacting with the user and calling appropriate methods for further
 * actions.
 *
 */
public class PlayRisk {

	public static void main(String[] args) {
		PlayRisk game = new PlayRisk();
		System.out.println("Welcome to Risk Game");
		System.out.println("To continue, select a map from the below mentioned existing maps or
create a new one.");
		game.printMapNames();

		//read first command
		Scanner read = new Scanner(System.in);
		String command = read.nextLine();
		Command.Phase gamePhase = Command.Phase.NULL;	//maintains phase of the game
		Command cmd = new Command();
		gamePhase = cmd.parseCommand(null, command);
		while(gamePhase!= Command.Phase.REINFORCEMENT) {
			command = read.nextLine();
			gamePhase = cmd.parseCommand(null, command);
		}
```

Naming Conventions:

- Class and interface names are **<u>CamelCase</u>** and it is recommended to use the whole word and avoid acronyms/abbreviations. For example **<u>class Raster</u>** or **<u>class ImageSprite</u>**
- **Package —** names com.deepspace over com.deepSpace or com.deep_space
- **File** — names are CamelCase and end with .java matching the class name. There is one public class per file with each top-level class in its file
- **Method** — names should be verbs in mixed case with each internal word capitalized for example run(); or runFast();
- **Constants** — should be uppercase with "_" separating each words for example int MIN_WIDTH = 44; and int MAX_WIDTH = 99;
- **Variable** — a name that tells the reader of the program what the variable represents i.e. if you are storing a test grade then pick grade vs var1 . Keep the variable names short avoid including metadata.

```
public class Command {

 public static boolean allArmiesPlaced = false;

   public GameMap map;
   public RunCommand runCmd;
   public StartUp startUp;
   public Reinforcement rfc;
   public Fortification ftf;
   public enum Phase {NULL, EDITMAP, STARTUP, ARMYALLOCATION,
REINFORCEMENT, FORTIFICATION, TURNEND, QUIT};
   Phase gamePhase;
   public ArrayList<Player> players;

   public Command() {
      map = new GameMap();
      runCmd = new RunCommand();
      startUp = new StartUp();
      rfc = new Reinforcement();
      ftf = new Fortification();
      players = new ArrayList<Player>();
      gamePhase = Phase.NULL;
   }
```

Prefer & Avoid:

Formatting and indentation are all about organizing your code to make it easy to read, and it includes spacing, line length, wraps and breaks and so on

- **Indentation** — Use *2 or 4 spaces* and stay consistent
- **Line length** — Up to 70 to 120 characters depending on affect on readability. It's important to eliminate the need for horizontal scrolling and place line breaks after a comma and operator.

```
while(gamePhase!=Command.Phase.TURNEND) {
  command = read.nextLine();
  gamePhase = cmd.parseCommand(p, command);
}
gamePhase = Command.Phase.REINFORCEMENT;
cmd.setGamePhase(gamePhase);
traversalCounter++;
```

If-checks —Writing well-formatted code makes it easy to spot typos and errors to the author and the code reviewers, see below:

```
if (data[1] != null) {
        if (this.isMapNameValid(data[1])) {
          System.out.println("In loadmap: " + data[1]);
           mapName = data[1];
           map = runCmd.loadMap(mapName);

           if (map != null) {
              if (!map.getValid()) {
                 System.out.println("Map is not valid for game play");
                 gamePhase = Phase.NULL; // map is not valid for game play. so return to NULL Phase
              } else {
                 gamePhase = Phase.STARTUP; // startup phase of game started from here
              }
           } else {
              gamePhase = Phase.NULL;
           }
           break;
        } else {
           System.out.println("Map name not valid.");
        }
```

```
    } else {
       System.out.println("Empty Name");
    }
```

Switch case:

- Always have a default case even without code.
- Use /* falls through */ to indicate the control falls to next case.

```
switch (condition) {
  case ABC:
    statements;
  /* falls through */
  case DEF:
    statements;
    break;
  default:
    statements;
     break;
}
```

Declarations and Assignments:

- One declaration per line is recommended since it encourages comments as shown below.

```
public Phase parseCommand(Player player, String newCommand) {

    String mapName = null;
    String continentName = null;
    String countryName = null;
    String neighborCountryName = null;
    String playerName = null;
    String fromCountry = null;
    String toCountry = null;

    int controlValue = 0;
    int numberOfArmies = 0;
    int armiesToFortify = 0;
```