PE1 Cheatsheet

💡 General Reminders

- **Write out** all required classes, fields, and methods (on a separate piece of paper if req) before starting. Spend decent time on this!
- Also check the **required outputs** CAREFULLY.

▼ Setup

```
set hlsearch
set incsearch
set number

set background=dark
color gruvbox
```

▼ Bash

```
# delete swap files
find . -name "*.swp" -type f
find . -name "*.swp" -type f -delete
```

▼ 🔍 Testing

```
# always remember:
rm -rf OUT

# Run with inputs and redirect output
java Ex2 < inputs/Ex2.1.in > OUT

# Check for differences
vim -d OUT outputs/Ex2.1.out
```

▼ VIM

> 💡 **selected commands only.**
>
> - *for more: https://vim.rtorr.com/*

`.` - repeat last command

▼ Undo/Redo

`u` - undo | `Ctrl` + `r` - redo

`Ctrl` + `u` - Delete all newly added characters in current line (`INSERT` mode)

▼ Insert

`e` - basically `x` then `i`

`i` - before cursor

`I` - **beginning** of line | `A` - **end** of line

`o` - new line **below** | `O` - new line **above**

Can also exit using `Ctrl` + `C`

▼ Search *(and Replace)*

`n` - next occurence

Replace: `:%s/ <search-phrase> / <replace-phrase> / options`

- Select words, replace all with check: `:%s/ \< search \> / replace /gc`

▼ Registers, Deleting, Yanking

`:reg` - show registers *(only lines are saved in the history)*

💡 Note that you can also do `yw` to yank a word

use above with `:" x p` to paste contents of register `x`, eg `:"3p`.

`p` - paste **after** cursor | `P` - paste **before** cursor

▼ Main Java

Method Signature: Only considers the parameters: name of method, number, type, order of params - ❗ **NAME OF PARAMS**, Return type etc are NOT part of the signature.

- 🚧 Note that **overloading** and **overriding** are different:
  - **Overloading**: same name, different descriptor. | **Overriding**: same descriptor/signature.

`Final`

- Class - cannot be inherited
- Method - cannot be overwritten
- Field - cannot be reassigned

`Static`

- Fields: Class fields
- Methods:
  - Should be accessed through the class: eg `Circle.getPi()` rather than say, `c1.getPi()`.

`Interface`

- NOTE that something like `I i2 = (I) new A();` would compile regardless of whether A actually implements I - the compiler thinks there is a possibility that A implements I.
- All methods in an interface are `public abstract` by default.

Generics/Wildcards

`class < T implements Comp< T > extends Object > A {...}`

`public static < S > boolean contains( Seq<? extends S > , S obj) {…}`

- Note that you cannot access non-static (instance) fields from a static method

When assigning, it's a ***widening type conversion***.

`Abstract`

- cannot be instantiated
- Only **one method needs to be** `abstract` - no need body:
  - `abstract public int foo();`

Regarding Inheritance

https://docs.oracle.com/javase/tutorial/java/landl/subclasses.html

- `private` fields cannot are not accessible in `child classes`. Use public getters.

```
@SuppressWarning("unchecked");
T[] arr = (T[]) new Object(size);

# NO RAW TYPES:
Seq s = new Seq(size);
# Yes:
Seq<int> s = new Seq<int>(size);
```

Can do (with wildcards):

- `A instanceof A < ? >` - can't do `... A <String>`
- `new Comparable < ? > [10];`

Always `@Override` !

▼ ⚠🚧 ❗ Exceptions ❗ 🚧⚠

Exceptions Notes

Unchecked exceptions <: `RuntimeException`

- not explicitly caught or thrown

Checked exceptions <: `Exception` (only)

- Can be thrown even if code is written perfectly, eg *input issue*

> 💡 the whole sequence from throw to catch (in terms of methods) needs to have the *exception* ***explicitly thrown***.

▼ Custom Exception

```
class CustomException extends Exception {
    CustomException(String msg) { super(msg); }

    @Override
    public string toString() { return ""; }
}
```

The return type of an **overriding** method **CANNOT be a supertype** of the overriden return type.

- *noteworthy when it comes to* ***type erasure*** *and related types with generics but* ***not bounded correctly****.*

Type Checking (Example)

> Unchecked warning - compiler unsure if line is type safe - using Raw Type, compiler cannot check it is is safe to pass a (`Integer` to the `keep` method)

> Runtime: program assigns value 123 to `x` - allowed due to type erasure: `T x` becomes `Object x` after type erasure.

> **ClassCastException** as it tries to cast an `Integer` to a `String`