Object-Oriented-Programming Basics cheatsheet

Objects

- **Objects** consist of data along with the procedures that operate on that data.
  - We call this data **fields** or **instance variables**.
  - We call the procedures **methods** or **operations**.
  - When we call the method of an object, we say that we **make a request** or **pass a message** to the object.
- Object data is can only be altered by calling the object's methods. This is called **encapsulation**.
- If the data is declared to be **private**, then it cannot be seen from outside the class, thus protecting these internal implementation details from leaking out. This is called **information hiding**.

Types

- Every operation is characterized by a name, the kinds of objects it takes as parameters, and the kind of value it returns. These elements collectively are the **signature** of the operation.
- The selection of all the signatures for the operations that an object can handle is the object's **interface**.
- A collection of signatures is called a **type**. We say that **an object *has* a type** if it has an implementation for each operation specified in the type.
- In Java we can express such types via the formal element called a **Java Interface**.

  Interfaces specify *what* an object can do, and NOT *how* to do it. The latter is the job of classes.

- Objects can have the same interface, but different implementations. A call like o.draw() sends the draw message to the o object. Therefore o must implement an interface that contains a draw method, but we don't know what specific implementation of draw is executed until runtime. At runtime, the implementation of draw that the specific object o has will be executed. This is known as **dynamic binding**.
- Dynamic binding allows us to write programs based on an object's interface, then swap objects at runtime, as long as they have the same interface. This allows us to vary the implementation without changing the code, so that the result of o.draw() can vary depending on which specific method is executed. This ability to substitute an object with a given interface for another object with the same interface is called **polymorphism**.

Classes

- A **class** specifies the internal data and method implementations for objects.
- We create objects by **instantiating** a class. We then call the object an **instance** of the class.
- An **abstract class** contains methods that are possibly not implemented, but simply declared to have a specific signature. Abstract classes cannot be instantiated. They can thus be used to express a *type*.
- **Concrete classes** contain implementations of any abstract methods from the abstract classes, and they may also **override** implementations provided by the parent class.

Variables

Methods and objects work with a number of different "variable" symbols. They vary in their **scope**, i.e. the specification of all the parts of the code where they exist.

- **Instance variables**, also called **fields**, are unique to each object, typically created when the object is instantiated. Their values are shared amongst all the methods of the object.
- **Static variables** or **static fields** are properties associated with a class and are shared amongst all object instances of that class. Similarly static methods can be called just using the class name and without requiring a class instance.
- **Parameters** or **arguments** are passed to the method from its caller. Their value only extends to the end of the specific function call.
- **Local variables** or simply **variables** are defined within a function and exist only within the innermost set of curly braces that contains their declaration.