More on Objects in JAVA™

Inheritance :

Definition: A subclass is a class that extends another class. A subclass inherits state and behavior from all of its ancestors. The term "superclass" refers to a class's direct ancestor as well as to all of its ascendant classes.

A subclass inherits all of the members in its superclass that are accessible to that subclass unless the subclass explicitly hides a member variable or overrides a method. Note that constructors are not members and are not inherited by subclasses.

For example, all classes are descendents of the *Object* class. This *superclass* has a method called `toString()` which is the method called by default when you attempt to `print` an object. It would normally print a representation of the *reference*. However, you could override it to do what ever you wish.

Access :

`public`

- A public class can be accessed by any code that can access the package in which the class is declared.
- A public class member (variable, method or constructor) can be accessed by any code that can access the class in which it is declared.
- A public member can be accessed directly from a subclass without qualification.
- A public constructor can be accessed from a subclass by the expression `super()`.

`private`

- A class member (variable, method or constructor) declared as private can be accessed only from within the compilation unit in which it is declared.

---

from the *Java Master Reference*

`protected`

- The protected constructor of a class can be accessed by code in the same package in which the class is declared, or by code in any subclass of the class in which the protected constructor is declared.

- A protected member(variable, method, constructor) can be accessed directly from a subclass without qualification.

- A protected constructor can be accessed from a subclass by the expression `super()`.

- A protected variable or method is inherited by any subclass of the class in which it is declared.

default

- A class with no access parameters can only be accessed from within the package in which it is declared.

- Members (methods and variables) default to `public`.

- A constructor can be accessed from a subclass by the expression `super()` only if the subclass is in the same package.

- A variable or method with default access is inherited only by subclasses that are in the same package as the class in which it is declared.

Final :

Nothing further can be done to modify or extend anything that is declared `final`.

- `final class`:
  A `final class` cannot be subclassed. Using this operator gives the compiler more freedom for optimizations.

- `final` method:
  A `final` method cannot be overriden. Note that in a `final class`, all methods are considered `final`.

- `final` variable:
  A `final` variable is a constant. The variable declaration must have an initial value. From there on, this value cannot be changed.

Static :

- `static` field:
  There will never be more then one copy of a static field. This field is considered the property of the class and not of any specific instance. Concretely, all objects from a class will always have the same value inside a static variable.

- `static` method:
  A `static` method is part of the class, not of an instance. A static method can only refer to other static entities of the class

Abstract :

Used to model an abstract concept without allowing the user to create instances of it.For example, the Number class in the `java.lang` package represents the abstract concept of numbers. It makes sense to model numbers in a program, but it doesn't make sense to create a generic number object. Instead, the Number class makes sense only as a superclass to classes like Integer and Float, both of which implement specific kinds of numbers.

Exceptions :

The exception is the Java way of handling errors.

- Exceptions are objects.
- They propagate through a program either by a user programmed `throw` statement or by a generic error in the behavior of the program. For example, looking at the method `readLine()` of the class `BufferedReader` :

  ```
  public String readLine() throws IOException
  ```

  may meet a situation where an error needs to be generated. When this event will happen, the object of type `IOException` will be created, your method wil end and the object will be sent to the calling method. If the exception reaches the starting point of your program, this one crashes.
- The programmer may `catch` exceptions to manage errors himself. However, this is dangerous since it is not always obvious what steps the Virtual Machine will need to take when meeting a specific error. Blocking it from doing this job properly could lead to disastrous errors.
- There exists a special subset of Exceptions which cannot be detected at the compilation time, these can propagate through the program without any guidance (`throws`). These are subclasses of `RunTimeException`.