Effective Programming

# Creating & Destroying Objects

## Consider Static Factory Methods instead of constructors

A class can provide its clients with static factory methods **instead of**, or **in addition to**, public constructors.

* **Advantages:**

**One advantage** of static factory methods is that, unlike constructors, they have names. with a well-chosen name is easier to use and the resulting client code easier to read. for a probably prime number for example:

BigInteger(int, int, Random) vs BigInteger.probablePrime

A class can have only a single constructor with a given signature. Programmers have been known to get around this restriction by providing two constructors whose parameter lists differ only in the order of their parameter types. This is a really bad idea. Instead, you can use static factory methods with carefully-chosen names.

**A second advantage** of static factory methods is that, unlike constructors, they are not required to create a new object each time they’re invoked. This allows immutable classes to use preconstructed instances, or to cache instances as they’re constructed, and dispense them repeatedly to avoid creating unnecessary duplicate objects. The Boolean.valueOf(boolean) method illustrates this technique: it *never* creates an object**. It can greatly improve performance if equivalent objects are requested often**, especially if they are expensive to create. The ability of static factory methods to return the same object from repeated invocations, allows classes to maintain strict control over what instances exist at any time. Classes that do this are said to be ***instance-controlled****.* There are several reasons to write instance-controlled classes. Instance control allows a class to guarantee that it is a singleton or non-instantiable. Also, it allows an immutable value class to make the guarantee that no two equal instances exist: a.equals(b), if and only if a==b. This is the basis of the *Flyweight* pattern. **Enum types provide this guarantee**.

# Methods Common to All Objects

# Classes and Interfaces

## Minimize the Accessibility of Classes and Members

A well-designed component hides all its implementation details, cleanly separating its API from its implementation. **Components then communicate only through their APIs and are oblivious to each others’ inner workings**. This concept, known **as *information hiding*****or** ***encapsulation***, is a fundamental tenet of software design