Using Java Objects

Strings

- String is a built-in class type
- Sequences of Unicode characters
- String objects are immutable (there is no way of modifying the actual sequence of characters stored in it after it is set once)

Working with Strings examples

- Manipulating:
  - StringBuilder
- Parsing:
  - StringTokenizer, StreamTokenizer
  - Scanner
  - Pattern, Matcher
- Formatting:
  - String.format
  - System.out.printf
  - PrintWriter.printf
- Printing:
  - System.out.println

The StringBuilder Class

- Creates mutable objects
- To create a String object from a Stringbuilder object, toString() should be used
- See Eclipse for example

Parsing texts

- StringTokenizer
  - See Eclipse for example
- Scanner
  - See Eclipse for example

Formatting Strings

- Formatting strings: System.out.printf(...), PrintWriter.printf(...)
  - First argument is the format string
- Common conversion characters:
  - b - boolean
  - s - String
  - d - decimal integer
  - x - hex integer
  - f - decimal floating point
  - a - hex floating point
  - n - line separator
- Common flags:
  - left justify
  - + - signed
  - 0 - zero padded
  - (- negative numbers in parenthesis

Dates

- java.util.Date:
  - Most of the methods are deprecated
  - Bridge between the Calendar and DateFormat classes
  - Mutable date and time, to a millisecond
- java.util.Calendar:
  - Convert and manipulate dates and times
  - java.util.GregorianCalendar is a concrete subclass

- java.text.DateFormat:
    - Format dates
    - Can handle locales
    - java.text.SimpleDateFormat is a concrete subclass
- java.util.Locale:
    - Used with DateFormat and NumberFormat
    - Format dates, numbers, and currencies for specific locale

Formatting Dates

- SimpleDateFormat
- See example

Working with Dates and Calendars

- java.util.Calendar: abstract encapsulation of the object Date
    - Concrete implementation: java.util.GregorianCalendar
    - Getters and setters for date fields
- Calendar field access
    - Calendar.YEAR - identifies the year
    - Calendar.MONTH - identifies the month
    - Calendar.DAY_OF_MONTH - identifies the day
    - Calendar.HOUR - identifies the hour
    - Calendar.MINUTE - identifies the minute
    - Calendar.SECOND - identifies the second

Comparing and Identifying Objects

- Normal equality (==) operator compares reference identity
- The Object class defines the equals method:
    - Determines whether objects are to be considered equal
    - Conditions for equality are defined by class designer
- See eclipse

Hash Code

- During an execution of a Java application:
  - Whenever it is invoked on the same object more than once during an execution of a Java application, the hashCode method must consistently return the same integer, provided no information used in equals comparisons on the object is modified. This integer need not remain consistent from one execution of an application to another execution of the same application.
  - If two objects are equal according to the equals(Object) method, then calling the hashCode method on each of the two objects must produce the same integer result.
- By default, hashCode is implemented by converting the internal address of the object into an integer

Object Lifetime

- Object "lives" as long as someone references it
  - Space is reclaimed for the heap by the garbage collector

Object Creation

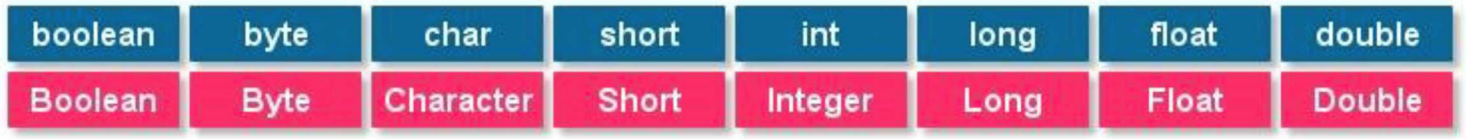
- Creating objects is expensive
- When an object is instantiated, the following steps occur:
  1. Memory is allocated.
  2. Instance fields are set to their default values.
  3. Any initialization blocks are called.
  4. The object's constructor is called.
  5. The object's super constructor is called.
  6. Superclass instance fields are given their specified values.
  7. The body of the superclass constructor is called.
- prefer primitives to their wrapper class counterparts
  - primitives perform always better than their corresponding wrappers

Garbage Collection

- The Java Virtual Machine maintains two sections of memory during the program execution: the stack and the heap.
- Automatic garbage collection:
  - Process of looking at heap memory
  - Identifying which objects are in use and which are not
  - Deleting the unused objects

- Unused object:
  - no longer referenced by any part of your program
- No keyword for deallocating memory
  - Deallocating memory is handled automatically by the garbage collector

Wrapper Classes



- Primitive types are not classes
- Primitive variables are not objects
  - They do not need to be instantiated
  - They do not have methods
- Wrappers give primitive object behaviour (immutable)
  - Used when primitives cannot be used (eg. in collection classes)
  - Conversion to and from wrapper objects is done by
    - Explicit conversion (Integer.valueOf(), Integer.intValue())
    - Autoboxing, -unboxing (since Java 5)

Autoboxing and Unboxing

- Autoboxing is the automatic conversion that the Java compiler makes between the primitive types and their corresponding object wrapper classes
- Unboxing is the other way around
- See eclipse

Enumerated Types

- Used with the enum keyword
- Typesafe enumeration of values
- Can be used in switch statements
- Enum declaration generates a new class behind the scenes

- The enum class body can include methods and other fields
- Some special methods when an enum is created
    - Static values method that returns an array containing all of the values of the enum in the order they are declared
    - Commonly used in combination with the for-each construct to iterate over the values of an enum type
- Constants defined inside Enum in Java are final you can safely compare them using "==" equality operator
- New collection classes EnumMap and EnumSet are added into collection package to support Java Enum
    - High performance implementation of Map and Set interfaces
- See eclipse