**AbstractCollectoin** (84)

**Adapter Class** (142,160)

**ArrayList**(86)

**Array Deque** (99)

**Box-class** (93)

**Class class** (108, 112)

**Clone** (94)

**ColorChooser** (162)

**Constructor class** (123)

**Dequeue** (81)

**Field Class** (119)

**FileChooser** (163)

**FunctionKey Buttons**(169)

**Graphics Diagram** (133)

**HashSet**(97)

**Icon** (150)

**Iterator Methods** (100)

**Inner/anon** (155)

**Inner class** (157)

**JFrame** (146)

**Label** (149)

**LayoutManager**(135) (layouts)

**LinkedHashSet**(98)

**LinkedList**(88)

**List** (101) (interface)

**Maps** (102)

**Map** (103) –interface

**Meta-programming** (96)

**Method Class** (117)

**Midterm Review** (76)

**Modal**(159)

**Modifiers Class** (115)

**MouseListener/Wheel**(160)

**Multi-Threading** (73)

**native** (122)

**Panel** (148)

**PriorityQueue** (91)

**Queue Interface** (79)

-demo(90)

**Reflection** (108)

**Review** (class/2classes in a file) (104)

**Standalone gui** (139)

**Swing** **Buttons** (171)

**Tooltip** (152)

**Thread** (144)

**TreeSet** (98)

**transient**(120)

**Type Descriptor** (109)

**XOR** (138)

.

Note:

-Class / Object (110)

-Graphics custom methods / Review (126)

-Simple Window (140)

-Gui kil to mil prog (141)

-alternative sto actionlistener (143)

-Foreground/contentPane/etc (167)

Adapter is a pattern that provides default (often empty) implementation of interface or abstract class. For example MouseAdapter provides empty implementation of MouseListener interface. It is useful because very often you do not really use all methods declared by interface, so implementing the interface directly is very verbose.

**import** java.util.HashMap;

**public** **class** SimpleName

{

**public** **static** **void** main(String args[])

{

String a = "hello";

HashMap<String, Object> d = **new** HashMap<>();

System.*out*.println(a.getClass().getName());

System.*out*.println(a.getClass().getSimpleName());

System.*out*.println(d.getClass().getName());

System.*out*.println(d.getClass().getSimpleName());

Class<?> c = String.**class**;

System.*out*.println(c);

//Class cc = "a".class; error

System.*out*.println(String.**class**.getName());

System.*out*.println(Integer.*TYPE*.isPrimitive());

}

}

java.lang.String

String

java.util.HashMap

HashMap

class java.lang.String

java.lang.String

true

call method public static void fixit(int x), if JE6 or more. Else use another method.

class Quiz1  
{  
    public static void main(String args[])  
    {  
         String str = System.getProperty("java.version");  
         int version = Integer.parseInt(str.substring(2,3));  
         if(version > 6) fixIt(60);  
         else  fixNum(60);       
    }  
  
    public static void fixIt(int x)  
    {  
         System.out.println("fixIt : " + x);  
    }  
    public static void fixNum(int x)  
    {  
         System.out.println("fixNum : " + x);  
    }  
      
}

Modifiers:

Class : public, abstract, final, private

Constructor : public, protected, private

Field : public, protected, private, static, final, volatile, transient

Method : public, protected, private, static, final, abstract, synchronized, native

Class         int Class.getModifiers()

Method int Method.getModifiers()

Constructor int Constructor.getModifiers()

Field  int Field.getModifiers()

Modifier      int Class.getModifiers()

 if(Modifier.PUBLIC ==mod)

   if(Modifier.STATIC ==mod)

Type Descriptor: Every class obj has a unique string rep we get via getName();

-regular classes : ex) java.lang.cloneable

-inner classes   : Main$InnerClass

-inner class XXX in YYY class : YYY$XXX

-inner class ZZZ in XXX class : YYY$XXX$ZZZ

BaseType Character  Type  Interpretation

B  byte      signed byte

C  char   Unicode character

D  double   double-precision floating-point value

F  float   single-precision floating-point value

I  int   integer

J  long   long integer

S  short   signed short

Z  boolean  true or false

[  reference  one array dimension

[[  reference  two array dimension

AnyClass we create in java is an object of the Class class\*\*\*\*

How to create a class object?

String a[]={"one","two","three"};

 // how to create an class instance.(1~4)

    Class c1=a.getClass();   // using getClass() method

    Class c2=boolean.class;   // appending 'class' to the class name.

    Class c3=Class.forName("[I");

    Class c4=java.lang.Integer.TYPE;

[[ Maps ]]  
collection that allows to store key-value entry.  
  
1. Maps do not implement Iterable interface.(no foreach loop)  
2. Do not support the Iterator() either.  
3. Then, what is available?  
   Map  
   Map.Entry  
   NavigableMap  
   SortedMap  
  
4. Map interfaces  
  
   interface Map<K,V>  
   void clear();  
   boolean containsKey(Object k)  
   boolean containsValue(Object v)  
   boolean equals(Object obj)  
   V get(Object k)  
   boolean isEmpty()  
   Set<K> keySet()  
   V put(K k, V v)  
   void putAll(map<? extends K, ? extends V> m)  
   Set <map.Entry<K,V>> entrySet()  
   V remove(Object...  
   int size()  
  
   interface SortedMap<K,V>  
   Comparator<? super K> comparator()  
   firstKey, lastKey(), headMap(K end), tailMap(K start), subMap(K start, K end)  
   K getkey()  
   V getValue()  
  
5. AbstractMap  
   EnumMap  extends AbstractMap  
   HashMap  extends AbstractMap  
   TreeMap  extends AbstractMap  
   WeakHashMap  extends AbstractMap  
   LinkedHashMap  extends HashMap  
   IdentityHashMap  extends AbstractMap

GUI NOTES : THREAD

A controversial subject in the industry is in regard to how to code applets and standalone programs. Many introductory books direct a particular usage of main for a standalone program and init for a JApplet.   
  
A professional program needs to make use of multi-threading. The main thread should be available for other activities and not used for our activities directly. Instead, your own created thread is in charge of drawing, etc. There should be at least two threads, the main thread and our thread. Otherwise, it is possible to see Swing programs go into an infinite pause. So to avoid that, we will always be using multithreading in our swing programs.   
  
A brief review and introduction of several related points:  
  
1) The entry points for programs:  
a. Standalone: main method  
b. JApplet: init method  
  
2) paint method in graphics programs (both standalone and JApplet) is a heavyweight process. It cannot handle the speed of repainting. You may recall, we also had to develop our own delay process, because the paint delay didn't work properly. Sun has made a lightweight version of paint for us to use, called paintComponent. Its usage is as follows:

3. paint  
instead of using paing use paintComponent;  
  
protected void paintComponent(Graphics g){  
    super.paintComponent(g);  
}

JFrame:

Standalone programs use JFrame, which consists of 3 layers:  
1) background layer, where graphics get drawn  
2) content pane layer, where GUI components are drawn  
3) glass layer, where tooltips and similar evanescent functions are drawn  
  
paintComponent(Graphics g): This method writes everything it deals with (graphics) to the background layer.  
  
JPanel is used for GUI components. If we can put the graphics in a GUI component, we can have the graphics and the GUI components in the same layer of the JFrame (i.e., the content pane) and thereby avoid conflicts such as overlaying positions.  
topic: Canvas  
Canvas is a container that allows us to put graphics into it or draw on it; and we can also add it to the content pane because it is a GUI component. There are two ways to create a Canvas: