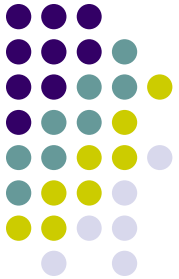
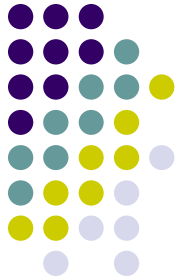
Java 200

4 - Advanced object oriented programming

Table of contents





SORTING / GENERICS

Generics

- A generic restricts the class or method to the use of a datatype
- A generic allows the method to return that datatype instead of the Object type.
- The replaceable variable for the datatype is indicated by the T and is enclosed in angle brackets

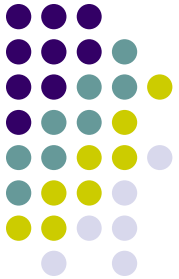
```
public class Person implements Comparable<T>{
    private String name;
```

```
public class Person implements Comparable<Person>{
    private String name;
```

Comparable interface

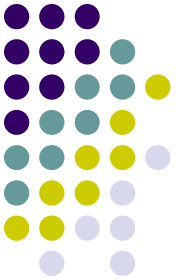
- Orders objects of each class that implements it.
- Natural ordering by the compareTo( ) (natural comparison method
- Lists and arrays are sorted automatically by Collections.sort and Arrays.sort). Keys in a sorted map or as elements in a sorted set
- One sorting method per class

compareTo( )



- x is sorted first before y when x.compareTo(y) >= 0
- x and y are not ordered when result is 0

compareTo( ) after generics



- This allows the datatype to be used throughout the scope of where T was defined.
- It will do the casting for you.

```
@Override
public int compareTo(Object o) {
    Person anotherPerson = (Person)o;
    return this.getName().compareTo(anotherPerson.getName());
}

@Override
public int compareTo(Person anotherPerson) {
    return this.getName().compareTo(anotherPerson.getName());
}
```

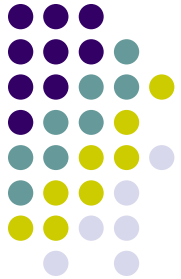
Exercise – SortablePerson (200)

- Create an array of people and sort them by name
  - Extend a Person class
  - Implement the Comparable interface
    - Use a generic to make it easier Comparable<SortablePerson>
  - Sort using a Person's name in compareTo( )
    - thisObject > otherObject ? 1 : (thisObject == otherObject ? 0 : -1)
    - Use compareToIgnoreCase( ) for better sorting
  - Arrays.sort(people)

Two-level sorting

- public int compare(SortablePerson o1, SortablePerson o2) {
- int firstSortResult = o1.getCity().compareTo(o2.getCity());
- if (firstSortResult != 0){
  - return firstSortResult; }
- return o1.getName().compareTo(o2.getName();
- }

Comparators



- A class which contains a compareTo() replacement method – compare( )
- Passed into sort method
- Create as many classes as you need ways to sort.
  - Save in same file if you don't make them public
  - Save with the entity file and provide a proxy method to use
- A function object

Exercise - SortablePerson

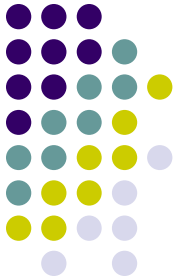
- Add another sorting method to reverse the array of people by name
  - Create a PersonReverseComparator class
  - Implement Comparator<Person>
  - Reverse the objects in the compare( ) method
  - Use the object to sort with
    - Arrays.sort(people,
      new ReversePersonComparator());

Sorting with lambdas

- Lambda = anonymous function, arrow function
- arg1 -> return value
- (arg1, …) -> return_value
- (arg1, …) -> function_call(arg1,…)
- (arg1, …) -> {statement; return var;}

- Arrays.*sort(people, (p1,p2)* -> p2.getName()
  .compareToIgnoreCase(p1.getName()));

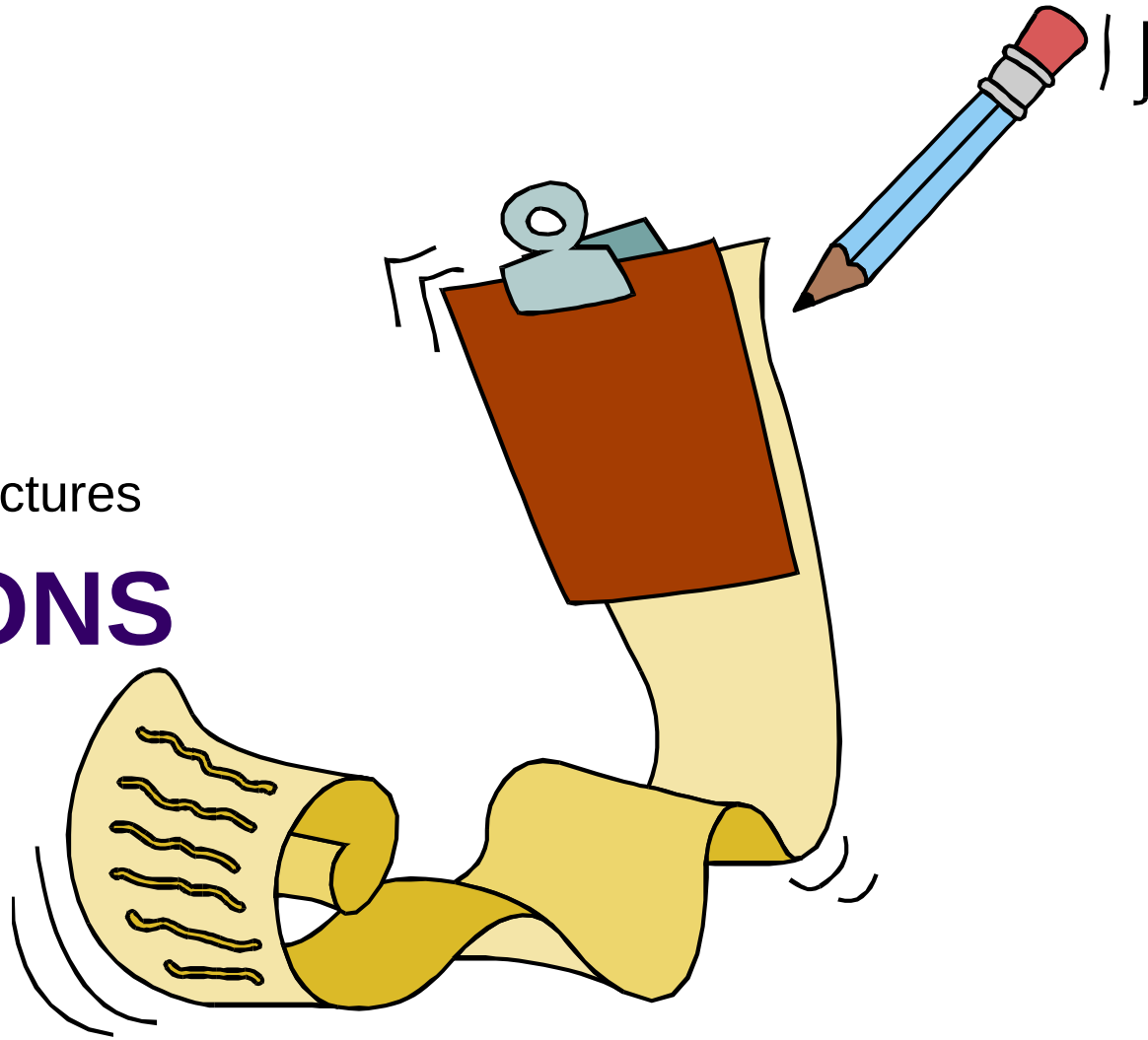
Sorting - summary



- Interface Comparable<T>
  - One algorithm in entity
- Comparator
  - Class for one algorithm each
- Inner class
  - Comparator implementation in entity
- Lambda
  - Anonymous function in entity

Object oriented data structures

COLLECTIONS

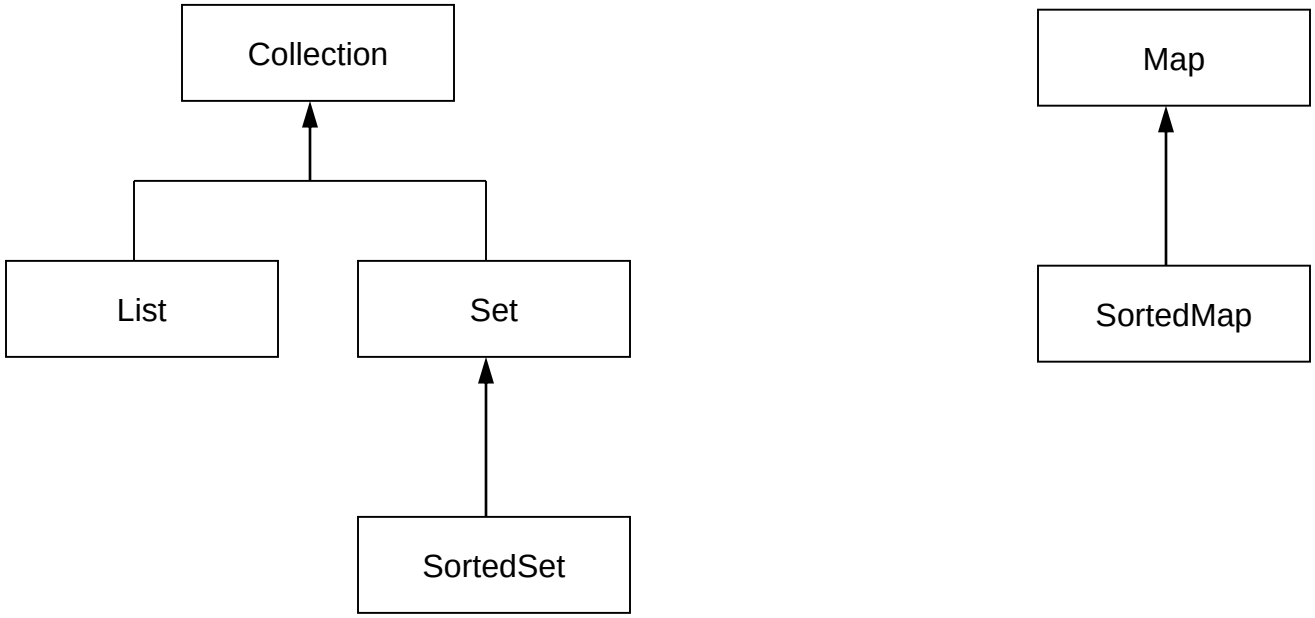


Collection vs. array

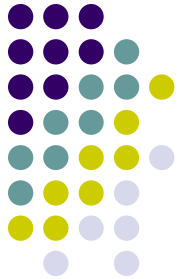
- A collection is an object
- An array is an object
- Arrays are fixed, collections are dynamically sized
- Arrays have limited functionality in Arrays
- Collections have more

Interfaces

- The collections framework
  - 1.2 – java.util



Other collections



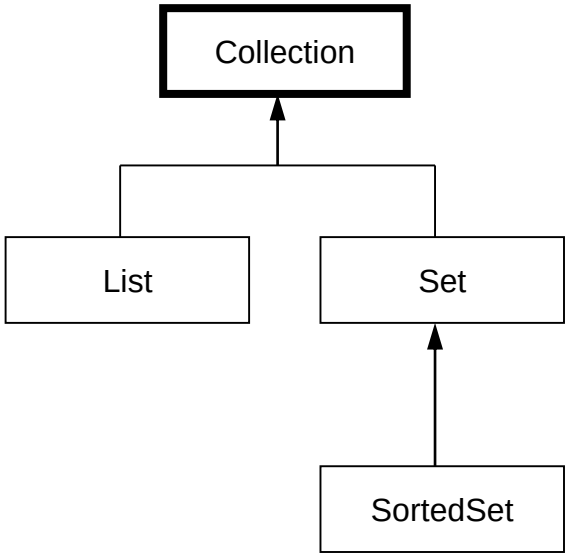
- java.util.Collections
  - utility functions for collections
- Collection
  - an interface that describes behavior for the one-dimensional structures

Four properties of data structures

Property	Description	Example
Sorted	Ascending order, sorted naturally using equals( )	Membership list
Ordered / unordered	Order is based on what position object is in.	Deck of cards
Duplicates allowed	Objects can have the same values	Books in a book store
Uses keys	An index is defined to refer to the object.	Unique social security numbers refer to people with equal names.

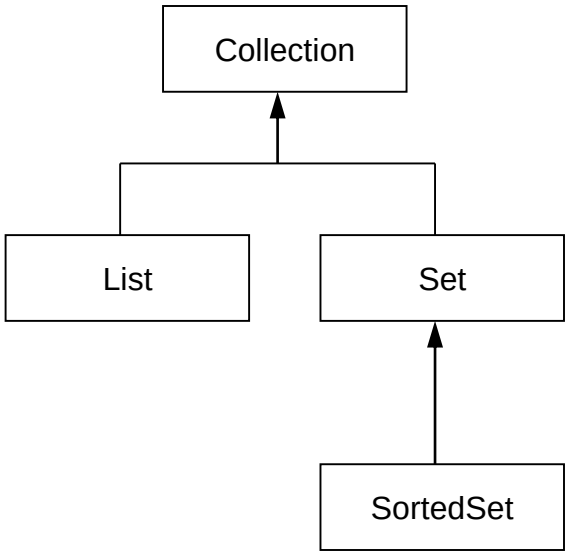
Collection superinterface

- Methods
  - `boolean add(Object element)`
  - `boolean remove(Object element)`
  - `boolean contains(Object element)`
  - `int size( )`
  - `boolean isEmpty( )`
  - `Iterator iterator( )`



Collection superinterface

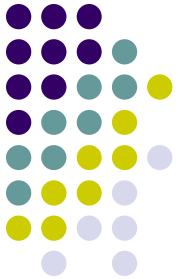
- Bulk methods
  - `boolean containsAll(Collection c)`
  - `boolean addAll(Collection c)`
  - `boolean removeAll(Collection c)`
  - `boolean retainAll(Collection c)`
  - `void clear( )`
- Bulk methods
  - `Object[] toArray( )`
  - `Object[] toArray(Object[] a)`



Examples

- `Collection anyCollection = new ArrayList( );`
- `int length = anyCollection.size( );`
- `anyCollection.add("new element");`
- `anyCollection.remove("old element that appears first");`
-
- `// toArray methods`
- `// Object[] values = anyCollection.toArray( ); // vague`
- `// String[] values = anyCollection.toArray( ); // error`
- `// String[] values = (String[])anyCollection.toArray( ); // error`
- `String[] values = (String[])anyCollection.toArray(new String[0]);`

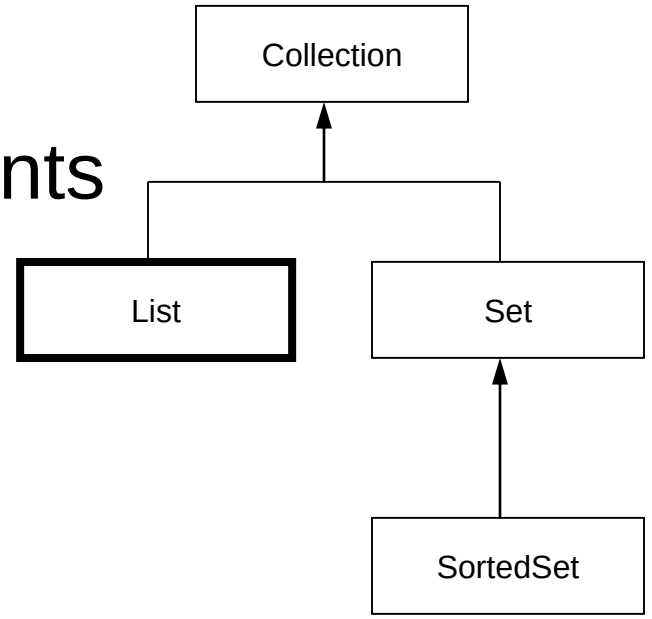
Exercise (CollectionTest)



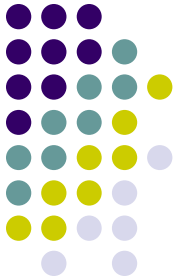
- `Collection<String> aCollection`
  `= new ArrayList<String>( );  // 1.7 syntax`
- `System.out.println(aCollection.add("dog"));`
- `System.out.println(aCollection.add("cat"));`
- `aCollection.add("bird");`
- `System.out.println(aCollection.remove("pig"));`
- `System.out.println(aCollection.remove("bird"));`
- `System.out.println(aCollection.add("dog"));`
- `System.out.println(aCollection);`

List interface

- an **ordered** collection
- can contain duplicate elements
- includes methods for
  - positional access
  - searching
  - list iteration
  - range-viewing

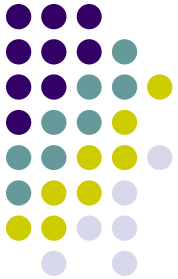


List interface



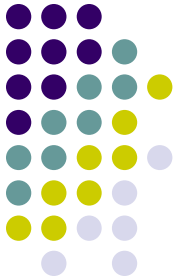
- Positioning
  - `Object get(int index)`
  - `Object set(int index, Object o)`
  - `void add(int index, Object o)`
  - `boolean addAll(int index, Collection c)`
  - `Object remove(int index)`
- Searching
  - `int indexOf(Object o)`
  - `int lastIndexOf(Object o)`

List interface



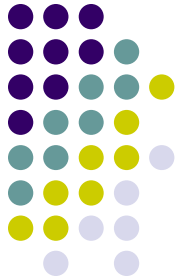
- Subset view – not a new object
  - `List subList(int from, int to)`
- Linked list
  - `addFirst(Object o)`
  - `addLast(Object o)`
- Concrete classes
  - Vector
  - **ArrayList**
  - LinkedList

Exercise (Boxing)



```
List<Integer> anArrayList
    = new ArrayList<Integer> ( );
for (int i = 0 ; i < 10 ; i++)
    //auto boxing
    anArrayList.add(i);
System.out.println(anArrayList);
    //auto unboxing
for (int integer : anArrayList) {
    System.out.println(integer + 100);
}
```

Exercise (LinkedListDigits)

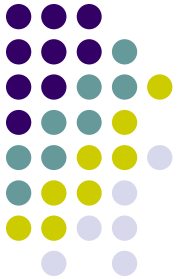


```
List<Integer> anArrayList = new ArrayList<Integer>( );
for (int i = 0 ; i < 10 ; i++) {
       anArrayList.add(i);
}
List aList = new LinkedList(anArrayList);
int length = aList.size( );
LinkedList aLinkedList = (LinkedList) aList;
System.out.println(aLinkedList.getFirst( ));
aLinkedList.addFirst(-1);
aLinkedList.addLast(10);

System.out.println(aLinkedList.removeLast( ));
System.out.println(aLinkedList);
```

Iterators

- Iterators
  - work on all Collection interface implementations.
  - are not array indexes.
  - track a position between elements and move with next( ) and previous( ).
- Create an iterator
  - Iterator<T> it = collectionName.iterator( );

Iterators

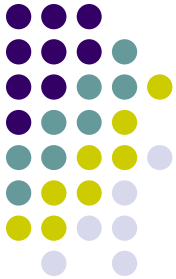


- Use an iterator
  - boolean hasNext( );
  - Object next( ); or <T> next( )
  - void remove( );

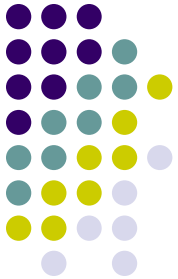
List iterators

- Extra methods
  - boolean hasPrevious( );
  - Object previous( );
  - int nextIndex( );
  - int previousIndex ( );
  - void set(Object o);
  - void add(Object o);

Exercises (IteratorLoops)



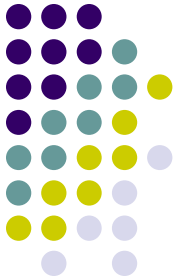
```
// add integers
for (int i = 0; i < 10; i++) {
    ints.add(i);
}
System.out.println("basic toString()");
System.out.println(ints);

Iterator<Integer> iterator = ints.iterator();
System.out.println("\n\nfor loop with iterator");
while (iterator.hasNext()) {
    System.out.print(iterator.next() + ", ");
}
```



Exercises

- Use a larger list
- String[] fontNamesInSystemArray = GraphicsEnvironment
  - .getLocalGraphicsEnvironment()
  - .getAvailableFontFamilyNames();
  - List<String> list = Arrays.asList(fontNamesInSystemArray);
- System.out.println(list);

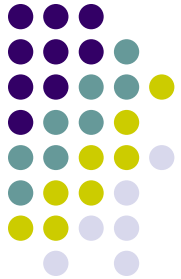
Iterator loop



- Using an iterator to print
- `Iterator<String> fontNamesIterator =`
  `SystemFontNames.list.iterator();`

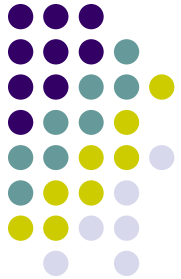
- `while(fontNamesIterator.hasNext()){`
- `System.out.println(fontNamesIterator`
  `.next());`
- `}`

Set interface



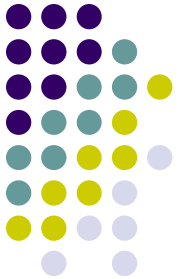
- a collection without duplicate elements
- dupes are ignored but no error or exception is generated.
  - make sure to override hashCode( ) and equals( ) for objects in a Set
  - Uses a comparison with more than just equals( )

Set interface



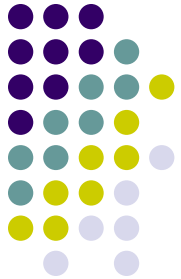
- Basic methods
  - int size( );
  - boolean isEmpty( );
  - boolean contains(Object o);
  - boolean add(Object o);
  - boolean remove(Object o);
  - Iterator iterator( );

Set interface



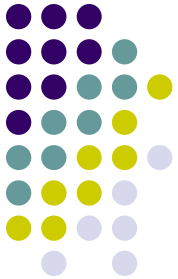
- Bulk methods
  - boolean containsAll(Collection c);
  - boolean addAll(Collection c);
  - boolean removeAll(Collection c);
  - boolean retainAll(Collection c);
  - void clear( );

Set interface



- Array methods
  - Object [ ] toArray( );
  - Object [ ] toArray(Object anArray[ ]);

Set interface



- Concrete classes: HashSet
  - order may not be constant
  - null is OK
  - find things fast
- Set<T> stuff = new HashSet<T>( );

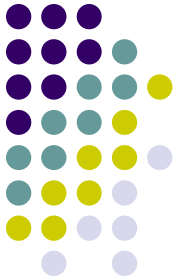
Exercise (CommandLineSet)

- Add command line arguments with duplicates and print the unique items.
- // convert array to List
- List<String> argList = Arrays.asList(args);

- // put args into a set
- Set<String> commandLineArgs = new HashSet<String>(argList);

- // print out set
- for (String arg : commandLineArgs) {
- 	System.out.println(arg);
- }

SortedSet interface



- Methods
  - Object first( )
  - Object last( )
  - Comparator comparator( )
- Subset methods
  - SortedSet headSet(Object to)
  - SortedSet subSet(Object from, Object to)
  - SortedSet tailSet(Object from)

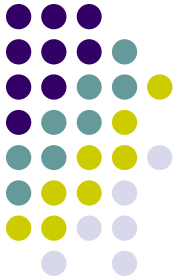
SortedSet interface

- Concrete class: TreeSet
- Constructors
  - TreeSet( )
  - TreeSet(Collection elements)
- Elements always sorted.
- All elements **must** implement Comparable.
- Insertion is faster than an array or linked list.

Exercise (CommandLineSortedSet)

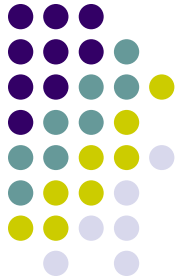
- // continued from previous
- System.out.println("---------- SortedSet");
- // put into a SortedSet
- SortedSet<String> aTreeSet
  = new TreeSet<String>(commandLineArgs);
- for (String arg : aTreeSet) {
- System.out.println(arg);
- }

Map interface



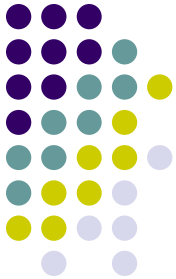
- A list of key/value pairs
- No duplicate keys
- Key can map to one value only
- Null can be a key
- Concrete classes: **HashMap**, TreeMap, Hashtable

Map interface



- Basic methods
  - Object put(Object key, Object value);
  - Object get(Object key);
  - Object remove(Object key);
  - boolean containsKey(Object key);
  - boolean containsValue(Object value);
  - int size( );
  - boolean isEmpty( );

Map interface

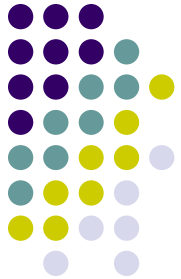


- Bulk methods
  - void putAll(Map m);
  - void clear( );
- Collection view methods
  - Set keySet( );
  - Collection values( );
  - Set entrySet( );
- An entry set is a Set of Map.Entry values

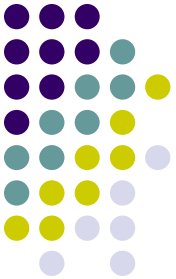
Map interface

- Concrete class: WeakHashMap
- Constructors
  - WeakHashMap( )
  - WeakHashMap(int initialCapacity)
  - WeakHashMap(int initialCapacity, float loadFactor)
- for debuggers to observe garbage collection behavior.

Map.Entry



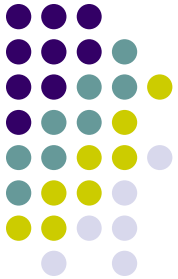
- An inner class – think of Map as a package name
- Methods
  - Object getKey( );
  - Object getValue( );
  - Object setValue(Object value);



Exercise

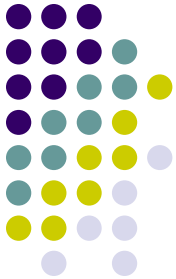
- Map<String,String> aHashMap
  = new HashMap< > ( );
- aHashMap.put("Game1","tic tac toe");
- aHashMap.put(null,"Chess");
- aHashMap.put("Game3","Checkers");
- aHashMap.put("Game3","Foosball");
- aHashMap.put("Game4","Chess");
- System.out.println(aHashMap);
- System.out.println(aHashMap.get("Game3"));

Exercise – with keyset



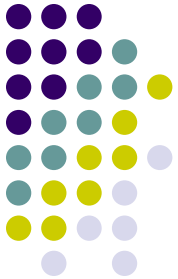
- `Set<String> keySet = aHashMap.keySet();`
- `for (String key : keySet) {`
- `    System.out.println(key + " = " + aHashMap.get(key));`
- `}`

Exercise – (MapEntryLoops)



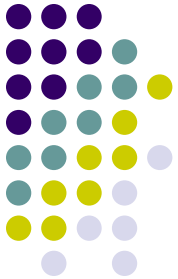
- // Recommended loop
- `for (Map.Entry<String,String> pair: aHashMap.entrySet()) {`
- `System.out.printf("Key: %s = Value: %s\n", pair.getKey(), pair.getValue());`
- `}`

SortedMap interface



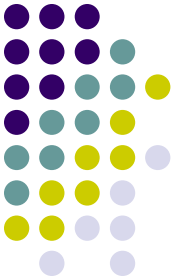
- Always sorted using a Comparator
- Basic methods
  - Object firstKey( )
  - Object lastKey( )
  - Comparator comparator( )
- Subset view methods
  - SortedMap subMap(Object fromKey, Object toKey)
  - SortedMap headMap(Object toKey)
  - SortedMap tailMap(Object fromKey)

SortedMap interface



- Concrete class: TreeMap
- Uses a compareTo( ) so items **must** implement the Comparable interface.
- no null keys
- Constructors
  - TreeMap( )
  - TreeMap(Comparator c)
  - TreeMap(Map m)
  - TreeMap(SortedMap m)

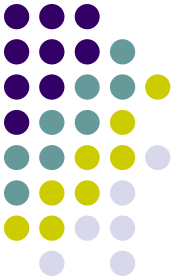
Exercise (SortedMapTest)



- `SortedMap<Integer,String> aTreeMap = new TreeMap<Integer,String>();`
- `aTreeMap.put(2,"tic tac toe");`
- `aTreeMap.put(3,"Checkers");`
- `aTreeMap.put(1,"Foosball");`
- `aTreeMap.put(4,"Chess");`
- `System.out.println(aTreeMap);`

- `SortedMap<Integer,String> partialView = aTreeMap.subMap(2,4);`
- `System.out.println(partialView);`

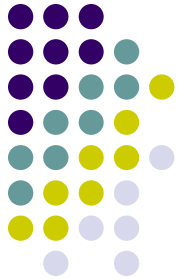
Exercise (SystemPropertiesMap)



- `SortedMap systemProps = new TreeMap(System.getProperties());`
- `System.out.println(systemProps);`

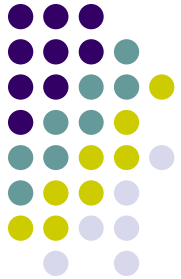
- `// sub map`
- `SortedMap systemPropsSubMap =`
- `systemProps.subMap("java.runtime.name", "java.vm.info");`

Collections class



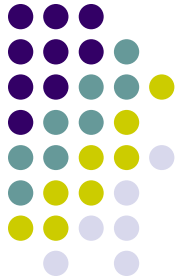
- Moving stuff around
  - static void copy(List dest, List src)
  - static void fill(List list, Object o)
  - static void reverse(List l)
  - static void shuffle(List list)
  - static void shuffle(List list, Random rnd)
  - static void sort(List list)
  - static void sort(List list, Comparator c)
  - static Comparator reverseOrder( )

Collections class



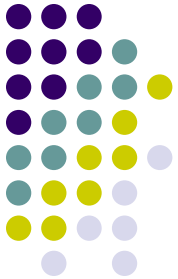
- Finding stuff
  - static int binarySearch(List list, Object key)
  - static int binarySearch(List list, Object key, Comparator c)
  - static Object max(Collection coll)
  - static Object max(Collection coll, Comparator comp)
  - static Object min(Collection coll)
  - static Object min(Collection coll, Comparator comp)

Collections class



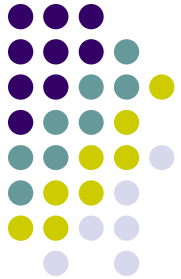
- Wrappers for thread-safe collections
  - static Collection synchronizedCollection(Collection c)
  - static List synchronizedList(List list)
  - static Map synchronizedMap(Map m)
  - static SortedMap synchronizedSortedMap(SortedMap m)
  - static Set synchronizedSet(Set s)
  - static SortedSet synchronizedSortedSet(SortedSet s)

Collections class



- Wrappers for unmodifiable views of collections
  - static Collection unmodifiableCollection(Collection c)
  - static List unmodifiableList(List list)
  - static Map unmodifiableMap(Map m)
  - static SortedMap unmodifiableSortedMap(SortedMap m)
  - static Set unmodifiableSet(Set s)
  - static SortedSet unmodifiableSortedSet(SortedSet s)

Collections class



- Other useful methods
  - static List nCopies(int n, Object o)
    - Stores only one object but with illusion of many objects. Efficient & immutable.
  - static Set singleton(Object o)
  - static List singletonList(Object o)
  - static Map singletonMap(Object key, Object value)

Exercise (MinAndMax)

- `List<Integer> aList = new ArrayList<Integer>( );`
- `Random randomGenerator = new Random( );`
- `for (int i = 0; i < 50 ; i++) {`
- `    aList.add(randomGenerator.nextInt(1000));`
- `}`
- `System.out.println("Minimum: " + Collections.min(aList));`
- `System.out.println("Maximum: " + Collections.max(aList));`

Arrays class

- Slow List initialization
  - `List<String> aList = new ArrayList<String>();`
  - `aList.add("Mon"); aList.add("Tue"); …`
- Fast List initialization
  - `String[ ] days = {"Mon", "Tue", "Wed", "Thu", "Fri", "Sat", "Sun"};`
  - `List<String> daysList = Arrays.asList (days);`

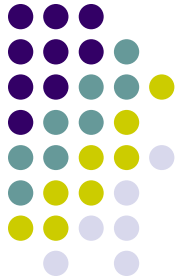
Properties & Preferences

- A property list is a map that stores things like environment variables or application .ini files.
- A map object of the Property class has three characteristics:
  - It stores keys and values as Strings.
  - It can be saved to a file and loaded from a file.
  - It contains a field of another Property object that stores defaults if needed.

Properties & Preferences

- Access to properties is through a Hashmap but uses Property methods
  - String getProperty(String key)
  - Enumeration propertyNames( )
    - The older Enumeration is used here because of the reliance on the legacy Hashmap.
  - Object setProperty(String key, String value)

Properties & Preferences

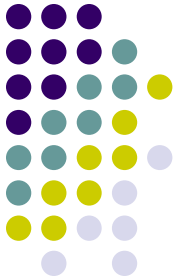


- Setting default values
  - Properties(Properties defaults)
  - String getProperty(String key, String defaultValue)

Properties & Preferences

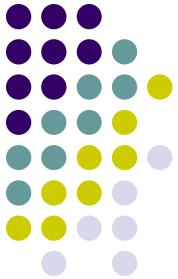
- Methods for I/O
  - void list(PrintStream out)
  - void list(PrintWriter out)
  - void load(InputStream inStream)
  - void store(OutputStream out, String header)
    - no defaults are written
    - optional header consists of a #, the header string and line separator
    - always a #, current date/time, and line separator

Exercise (SystemProps)



- Properties systemProperties = System.getProperties( );
- systemProperties.list(System.out);

Exercise (PropertiesFile)

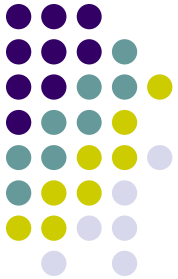


```
Properties settings = new Properties( );
settings.setProperty("FONT","Courier New");
settings.setProperty("BGCOLOR","HH0000");
try {
    FileOutputStream settingsOutStream = new
FileOutputStream("settings.properties");
    settings.store(settingsOutStream,"header comment");
    FileInputStream settingsInStream = new
FileInputStream("settings.properties");
    settings.load(settingsInStream);
} catch (IOException e) {
    System.out.println(e); }
System.out.println(settings + "\n");
settings.list(System.out);
```

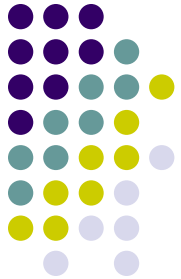
Early collections

- Vector
  - synchronized, prefer ArrayList
  - Constructors
    - Vector( )
    - Vector(Collection c)
    - Vector(int initialCapacity)
    - Vector(int initialCapacity, int capacityIncrement)

Early collections



- Hashtable
  - synchronized, prefer HashMap
- Enumeration
  - Iterator is better and includes remove( )
- Stack
  - LIFO, peek( ), pop( ), push(Object), empty( ), search(Object)
- BitSet
  - a Vector of boolean values



Lambda demo

- List<String> strings = Arrays.asList("a", "b", "c");

- static String result = "";
- strings.forEach((item) -> result += item);

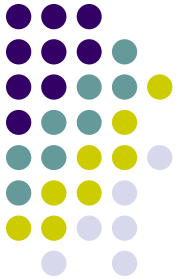
Exercise

- Create a deck of 52 cards, shuffle them, and deal and print a hand of five cards.
  - Dealer
    - public List<Card> getDeck()
    - public List<Card> deal(int numberOfCards, List<Card> deck)
    - public void shuffle(List<Card> deck)
    - main
  - Card
    - public static String[ ] suits, values (enums are better)
    - public static List<Card> getDeck() (static is better)

Summary

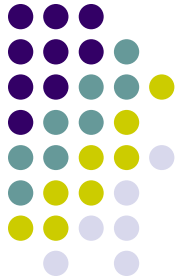
Interface(s)	Sorted	Ordered	Uses keys	Duplicates allowed	Null values OK
Collection, List	no	yes	no	yes	yes
LinkedList	no	yes	no	yes	yes
Set	no	no	no	no	yes, only one
SortedSet	yes	no	no	no	yes, only one
Map	no	no	yes	no keys, yes values	yes keys, yes values
SortedMap	yes, by key	no	yes	no keys, yes values	no keys, yes values

Summary



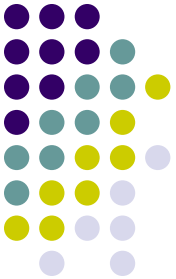
Interface	Abstract implementation	Concrete classes	Features
Collection	AbstractCollection		• add, remove, locate, select
List	AbstractList	ArrayList, *Vector*	• precise insertion
	AbstractSequentialList	LinkedList	• methods to use as a stack, queue, or double-ended queue
Set	AbstractSet	HashSet	• order fixed when created
SortedSet		TreeSet	• can be ordered with Comparator
Map	Dictionary	*Hashtable*	• key, value pairs; fixed order
	AbstractMap	HashMap	• like Hashtable
SortedMap		TreeMap	• can be ordered with Comparator

Summary



Traditional data structure	**Java implementation**	**types of uses**
ordered array	ArrayList	all purpose with searching
unordered array	any array	general purpose when number of items is known
linked list	LinkedList	Small quantity but inserting data in the middle often.
hash table	HashSet HashMap	Fast interaction with data. Spelling checkers, symbol tables, a person initiating data access.
binary search tree and balanced tree (red-black or 2-3-4)	TreeSet – red-black TreeMap	Consider first when arrays and linked lists are too slow. Guarantees performance even when the data isn't ordered.

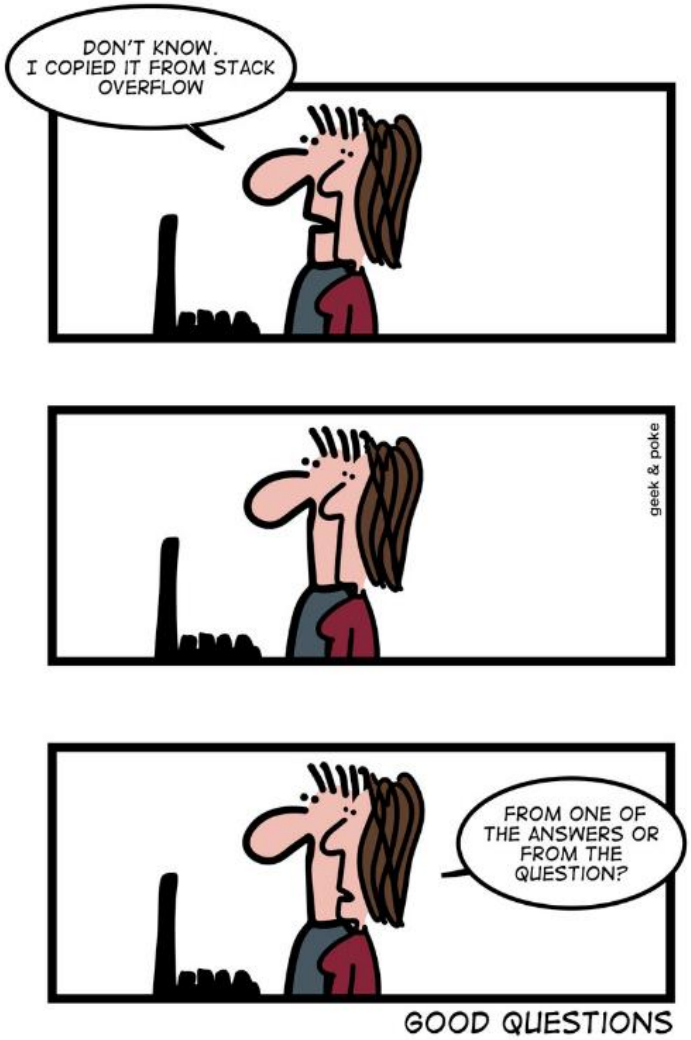
Summary - selection



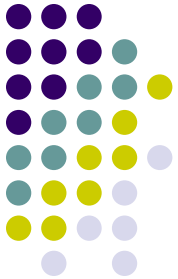
<table>
<tr><td rowspan="5">Do you have a small amount of data?</td><td rowspan="3">Yes – Is the amount of data predictable?</td><td rowspan="2">Yes – Is search speed more important than insertion speed?</td><td>Yes – ordered array</td></tr>
<tr><td>No – unordered array</td></tr>
<tr><td colspan="2">No - linked list</td></tr>
<tr><td rowspan="3">No - Does searching and insertion have to be very fast?</td><td colspan="2">Yes – hash table</td></tr>
<tr><td rowspan="2">No – Should key distribution be guaranteed to be random?</td><td>Yes – binary search tree</td></tr>
<tr><td>No – balanced tree</td></tr>
</table>

Redirect output, asserts, the debugger

DEBUGGING

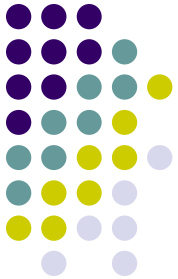


Old school



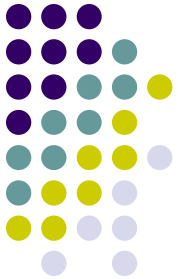
- code…
- // debug
- System.out.println(“x = ” + x);
- System.out.println(“y = ” + y);

Better – System.err



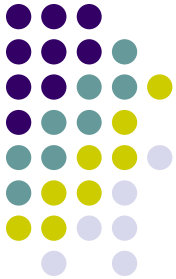
- code…
- // debug
- System.err.println(“x = ” + x);
- System.err.println(“y = ” + y);

Better yet – redirect to file



- Use internal I/O
  - FileOutputStream fos = new FileOutputStream("myLog.txt");
  - PrintStream ps = new PrintStream(fos);
  - System.setErr(ps);
- or
  - System.setErr(new PrintStream(new FileOutputStream("myLog.txt")));

Better yet – redirect to file

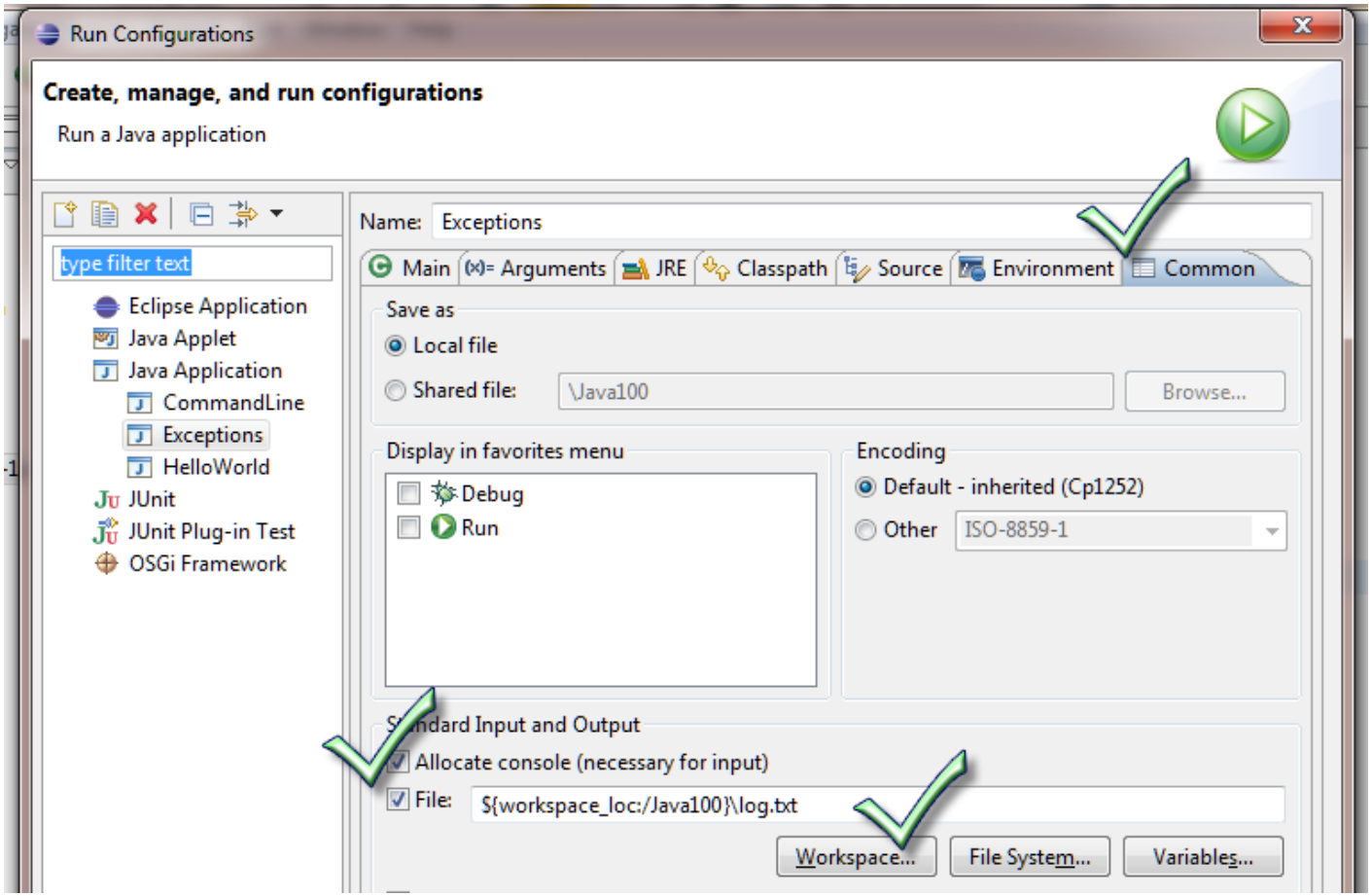


- Use OS redirection to file
- 1 = stdout, 2 = stderr
- Command line
  - java className 2> logFile.txt
  - java className 2>> logFile.txt
  - Sends to /users/<user>/ or /windows32 ?

  - java className 1>console.txt 2>error.txt

Redirect to file - Eclipse

- Send both stdout & stderr to file:

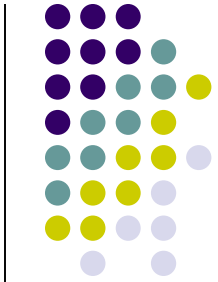


Even better - Logger

- Use a Java 6 Logger
- A file handler will reroute to a project level XML file.

```
private static final Logger LOG = Logger.getLogger(this class.class.getName());
static {
    try {
        FileHandler rerouter = new FileHandler("logger.txt");
        LOG.addHandler(rerouter);
    } catch (SecurityException e) { System.out.println(e);
    } catch (IOException e) { System.out.println(e);
    }
    LOG.info("Logging for " + this class.class.getName());
}
public static void main(String[] args) {
    LOG.warning("This is the end.");
}
```

The most common - Apache Log4J



- import org.apache.log4j.*;
- BasicConfigurator.configure();
  - Using a static block
- public static Logger log = Logger.getRootLogger( );
  - Logger.getLogger(“app.admin.users");
- log.setLevel(Level.WARN);
  - usually set in a config file

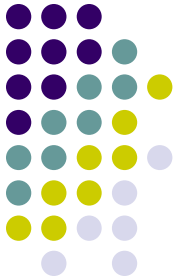
Apache Log4J

- Loggers
  - objects that transfer messages to output based on levels
- Appenders
  - directs ouput to file, console, etc. Additivity allows multiple outputs.
- Layouts
  - controls the output format to the appender
  - "%-4r [%t] %-5p %c %x - %m%n"

Apache Log4J

- *log.trace("This is for tracing.");*
- *log.debug("This is a debug message.");*
- *log.info("This is an information message.");*
- *log.warn("This is a warning.");*
- *log.error("This is an error error error.");*
- *log.fatal("Game over.");*

Apache Log4J



- PropertyConfigurator.configure(args[0]);

- Simpler alternative by same author
  - Simple Logging Facade for Java (SLF4J)

Assertions

- Not for logging, but confirmation of program state to validate requirements.
- assert **boolean expression** : “Error message”;

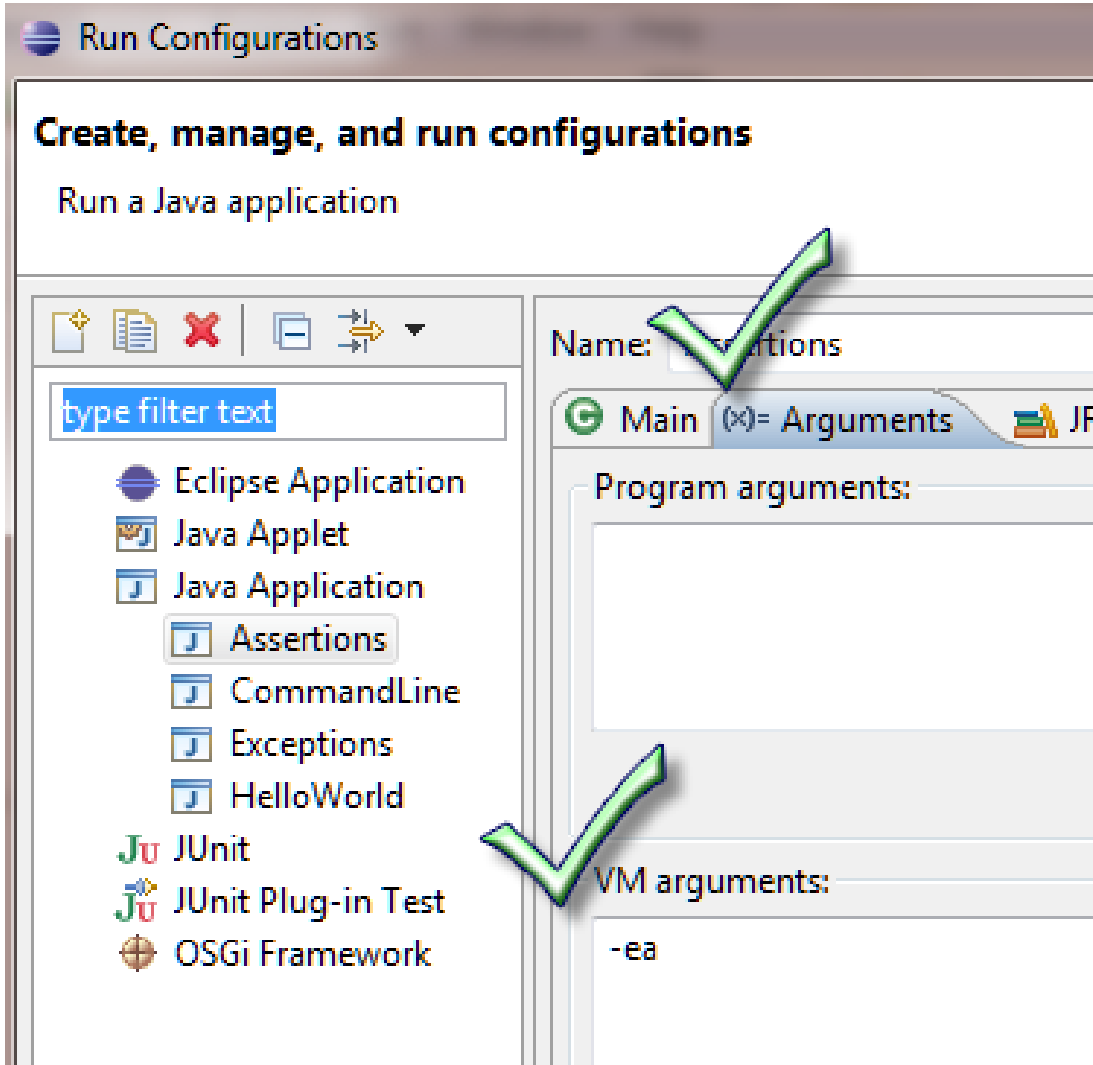
```
public static void main(String[] args) {
    assert args.length > 0: "No command args were entered.";
}
```

Assertions

- Used for
  - internal invariants – “this just has to be true…”
  - control flow invariant – “no way you can get here”
  - pre-condition – “state before running method”
    - check locks on threads
    - not for argument validity checks
  - post-condition – “state after running method to double check”
  - class invariant
    - run a method that performs a rule

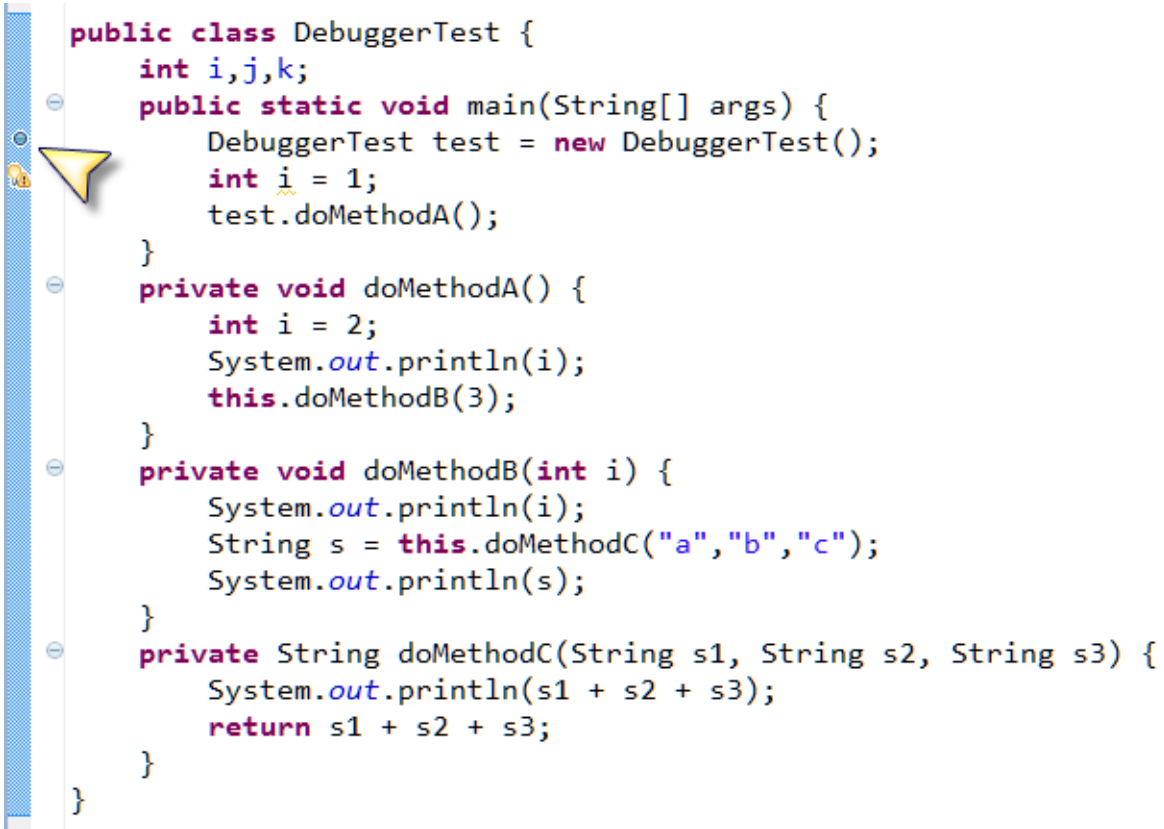
Enable assertions - Eclipse

- -ea



Debugger

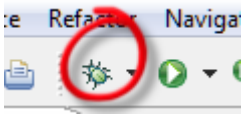
- Automated flow control and state of variables.
- Add breakpoint first. Double-click / click to add.



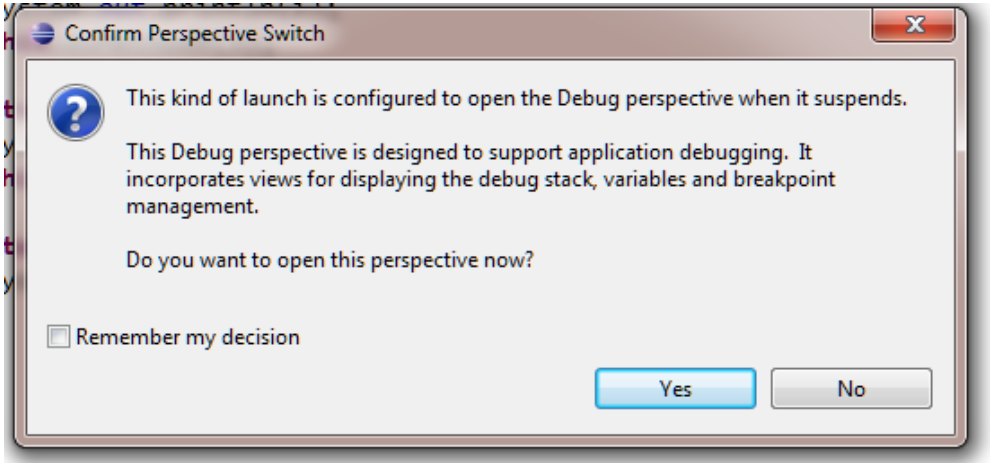
```
public class DebuggerTest {
    int i,j,k;
    public static void main(String[] args) {
        DebuggerTest test = new DebuggerTest();
        int i = 1;
        test.doMethodA();
    }
    private void doMethodA() {
        int i = 2;
        System.out.println(i);
        this.doMethodB(3);
    }
    private void doMethodB(int i) {
        System.out.println(i);
        String s = this.doMethodC("a","b","c");
        System.out.println(s);
    }
    private String doMethodC(String s1, String s2, String s3) {
        System.out.println(s1 + s2 + s3);
        return s1 + s2 + s3;
    }
}
```

Debugger

- Click on debug icon.

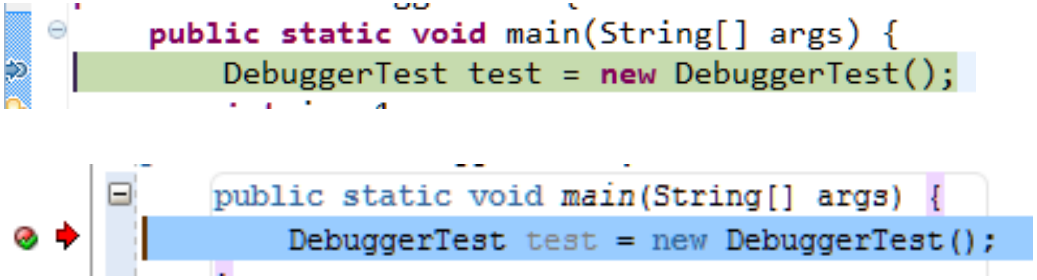


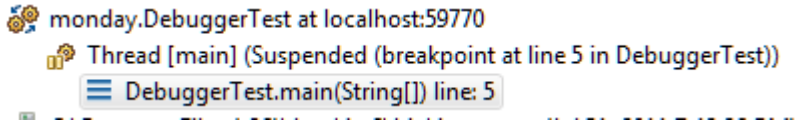
- Perspective change (Eclipse)
  - Yes
  - Remember my decision

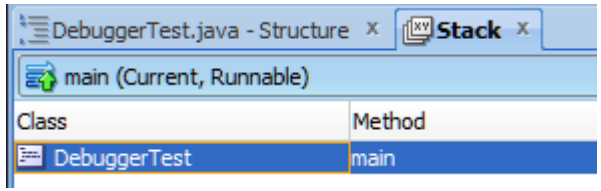


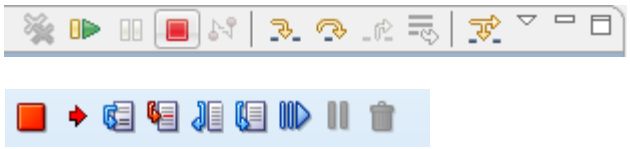
Debugger - features

- Pointer to and highlight current line to execute
- Stack trace for current Thread
  - first thread is always main
- Menu bar for debugging
  - Go to next breakpoint
  - Stop
  - Step…



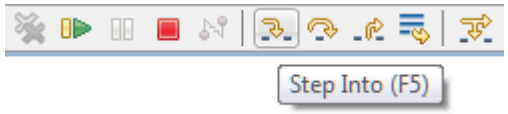


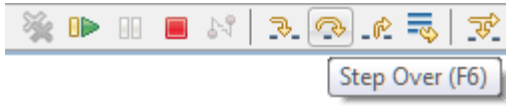


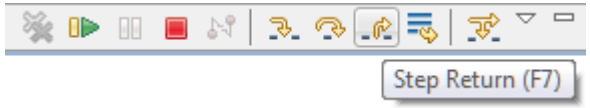


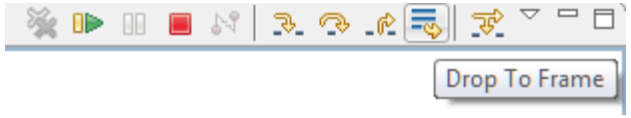
Debugger flow control - Eclipse

- Common step choices
  - Step Into
    - Will go to and step through a method being called.
  - Step Over
    - Will skip a method call
  - Step Return
    - Skips rest of method
  - Drop to frame
    - step backwards if possible



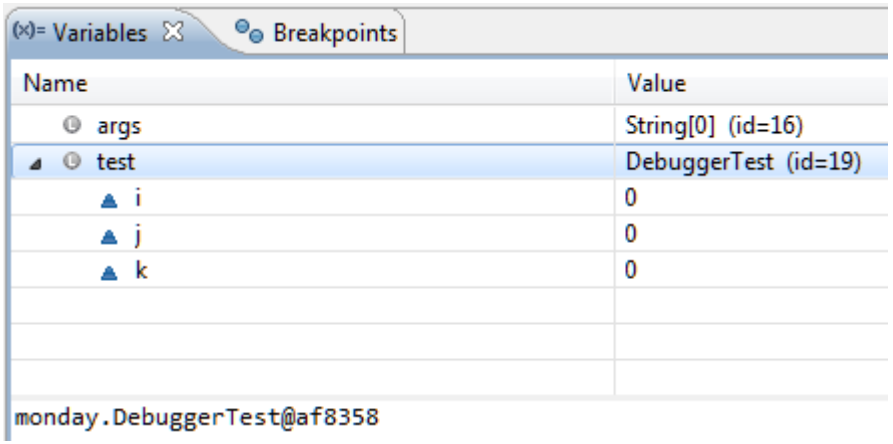


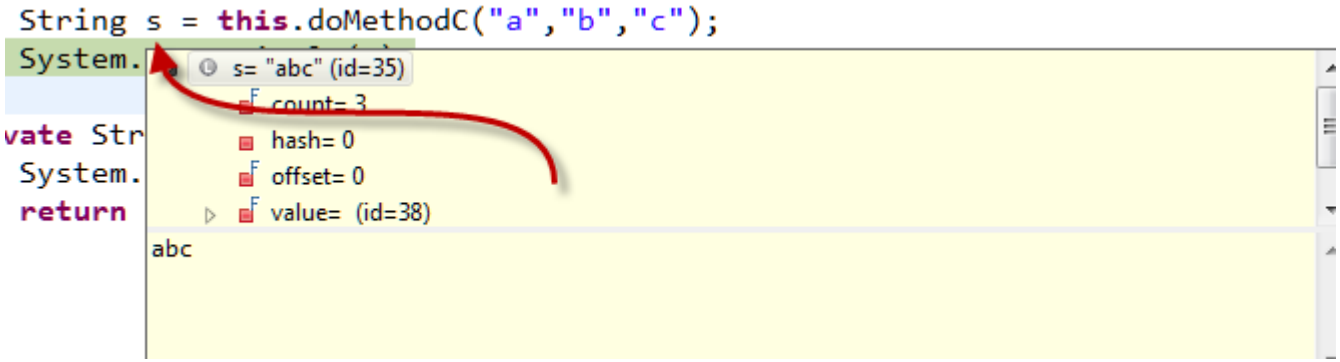




Debugger variables - Eclipse

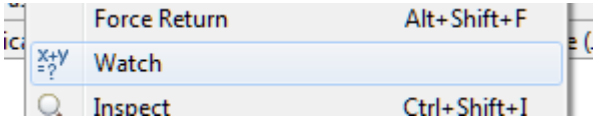
- Current class and local values
- Hover over a variable to show details

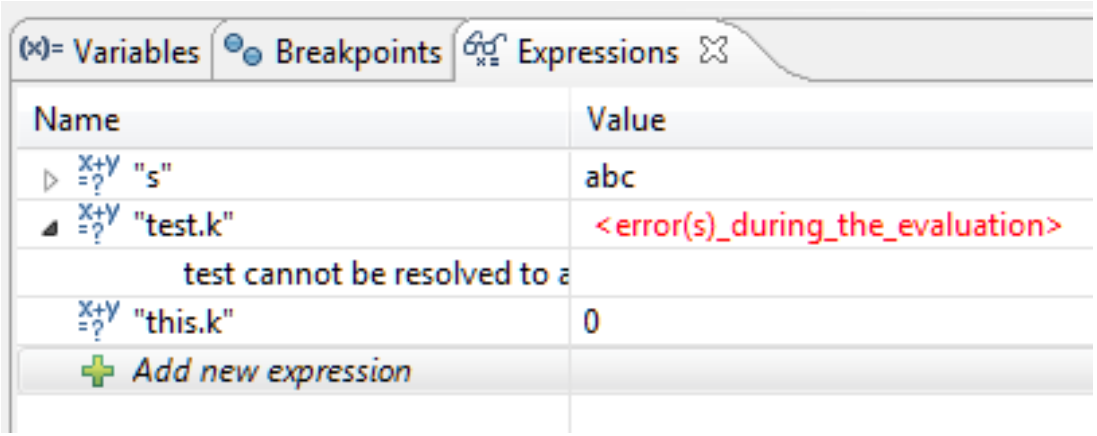




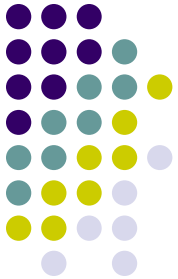
Debugger watch - Eclipse

- During a session, keep track of important variables even out of scope.
- Choose variable and right-click on Watch.





Conditional breakpoints

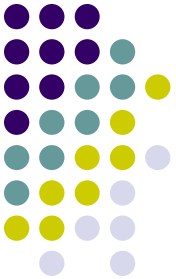


- i % 100

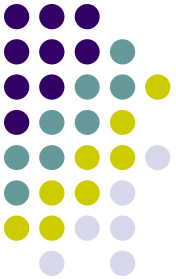
String cache, String methods, printf( ), Format, and searching

STRINGS

The String cache



- A cache stores/reuses previous values
- Objects created like primitives use it
  - String s1 = “abc”;
  - String s2 = “abc”;
  - System.out.println(s1 == s2);
- Objects created formally do not use it
  - String s3 = new String(“abc”);
  - String s4 = new String(“abc”);
  - System.out.println(s3 == s4);

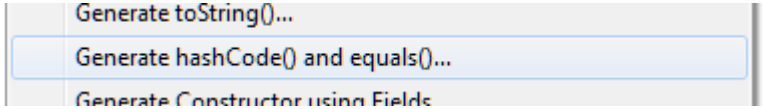


Exercise

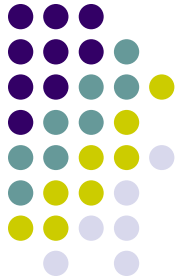
- String aString = "";
- long start = System.currentTimeMillis();
- for (int i = 0; i <= 10000000; i++) { // 10,000,000
- aString = "abc";
- System.out.print((i%100000==0)?".":""); } // 100,000
- long stop = System.currentTimeMillis();
- System.out.println("\nTime for String is: " + (stop-start));
- start = System.currentTimeMillis();
- for (int i = 0; i <= 10000000; i++) {
- aString = new String("abc");
- System.out.print((i%100000==0)?".":""); }
- stop = System.currentTimeMillis();
- System.out.println("\nTime for new String() is: " + (stop-start));

equals( )

- To compare two objects to see if their values are the same
  - fido.equals(rover)
  - “a string”.equals(“a string”);
  - s3.equals(s4);
- Create a hashCode() method at the same time
- Have Eclipse do it for you

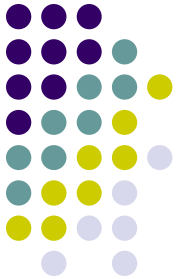


Comparing strings



- wrong
  - if(answer=="Yes");
- not so good - can throw a null exception
  - if (answer.equals("Yes"))
- good
  - if (answer!=null && answer.equals("Yes"))
- better
  - if (answer!=null && answer.trim().equalsIgnoreCase("Yes"))
- best
  - if (answer!=null && "Yes".equalsIgnoreCase(answer.trim()))

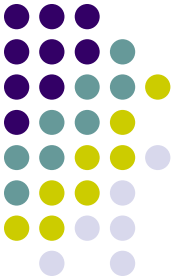
String methods



- charAt( )
- replace( )
- equals( ), equalsIgnoreCase( )
- startsWith( ), contains( ), endsWith( )
- length( )
- substring( ), split( )
- trim( )
- valueOf( )
- toLowerCase( ), toUpperCase( )

Expensive concat( )

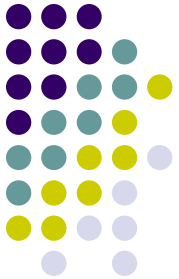
- Concat( ) is the same as +, +=
- Each concatenation creates a new object and uses processor time
- If you have > 10,000 or so concatenations, use StringBuffer
  - no new object is created
- StringBuilder (JDK 5) is a StringBuffer that is not synchronized and a little faster



Exercise

- String aString = "abc", bString = "def";
- StringBuilder sb = new StringBuilder("abc");
- long start = System.currentTimeMillis();
- for (int i = 0; i <= 100000; i++) {
- aString += bString;
- System.out.print((i%10000==0)?".":""); }
- long stop = System.currentTimeMillis();
- System.out.println("\nTime for String+String is: " + (stop-start));
- start = System.currentTimeMillis();
- for (int i = 0; i <= 100000; i++) {
- sb.append("def");
- System.out.print((i%10000==0)?".":""); }
- stop = System.currentTimeMillis();
- System.out.println("\nTime for StringBuilder is: " + (stop-start));

Exercise



- Modify the previous exercise to compare StringBuffer to StringBuilder

Format

- subclasses: DateFormat, MessageFormat, NumberFormat
- factory methods
  - getInstance( ) - default formatter
  - getInstance(Locale) - for the specified locale
  - specialized methods
- instance methods
  - format (Object)
  - parseObject(String)

MessageFormat – one use

- int planet = 7;
  String event = "a disturbance in the Force";
- String result = MessageFormat.format("At {1,time, short} on {1,date, long}, there was {2} on planet {0,number,integer}. Book price: {3,number,currency}",
  planet, new Date(), event, 12.9090909);
- The output is: At 12:30 PM on Jul 3, 2053, there was a disturbance in the Force on planet 7. Book price: $12.91

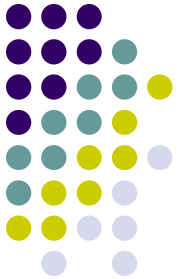
MessageFormat - reuse

- MessageFormat **form** = new MessageFormat("The disk \"{1}\" contains {0} file(s).");
- int fileCount = 1273;
  String diskName = "MyDisk";
- Object[ ] **testArgs** = { new Long(fileCount), diskName };
- System.out.println(**form**.format(**testArgs**));

MessageFormat - options

- String message =
  "Once upon a time ({1,date}, around about {1,time,short}), there was a humble developer named Geppetto who slaved for {0,number,integer} days with {2,number,percent} complete user requirements.“ ;
- Object[ ] variables =
  new Object[ ] { new Integer(4), new Date( ), new Double(0.21) }
  String output = MessageFormat.format(message, variables);
- System.out.println(output);

Single / plural / negation - simple



- String output =
  “Total: “ + i + “file” + (i==1 ? “” : “s”);
- // examples: There were some files.
- // There weren’t files.
- String output =
  “There were” + (hadFiles ? “ some” : “n’t any”) + “files.”

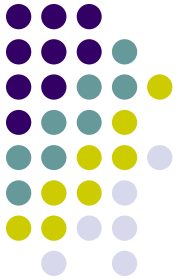
Single / plural - complex

- MessageFormat form = new MessageFormat("The disk \"{1}\" contains {0}.");
- double[] filelimits = {0,1,2};
- String[] filepart = {"no files","one file","{0,number} files"};
- ChoiceFormat fileform = new ChoiceFormat(filelimits, filepart);
- form.setFormatByArgumentIndex(0, fileform);
- int fileCount = 1273;
  String diskName = "MyDisk";
  Object[] testArgs = {new Long(fileCount), diskName};
- System.out.println(form.format(testArgs));

printf( ) (1.5)

- Similar to java.util.Formatter class (see docs)
- Uses a formatting string and then arguments to the string.
  - `System.out.printf("%s  %.4f  %,d", "a string %n", 2.0/3, 12345);`
  - `Calendar c = Calendar.getInstance( );`
  - `System.out.printf("%n%tr %n%tc %n%tl:%tM" , c, c, c, c);`

java.util.Formatter



- StringBuilder sb = new StringBuilder();
- Formatter formatter = new Formatter(sb, Locale.US);
- formatter.format("%4$2s %3$2s %2$2s %1$2s", "a", "b", "c", "d");

- String.format("Duke's Birthday: %1$tm %1$te,%1$tY", c);

Working with Date and Calendar ←GregorianCalendar

DATES & TIME

java.util.Date

- a point in time
- To get a current date
  - Date now = new Date( );
- To make a Date from a GregorianCalendar object:
  - Date turnOnTV = nextCSI.getTime();

java.text.DateFormat

- Create or parse strings of dates and times.
- Use a default DateFormat object to make a String out of a Date:
  - DateFormat tvFormat = DateFormat.getInstance();
  - String tvGuideItem = tvFormat.format(turnOnTV);
    // dd/mm/yy h:mm AM/PM

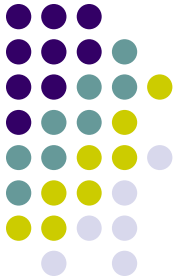
DateFormat factory methods

- use DateFormat constant types DEFAULT, SHORT, MEDIUM, LONG, & FULL with
  - DateFormat aDateFormat1 = DateFormat.getDateInstance(DateFormat.FULL);
  - DateFormat aDateFormat2 = DateFormat.getTimeInstance(DateFormat.SHORT);
  - DateFormat aDateFormat3 = DateFormat.getDateTimeInstance(DateFormat.MEDIUM , DateFormat.LONG);
- Using the DateFormat object
  - System.out.println(aDateFormat1.format(new Date() + " \n" + aDateFormat2.format(new Date() + " \n" + aDateFormat3.format(new Date())));

DateFormat examples

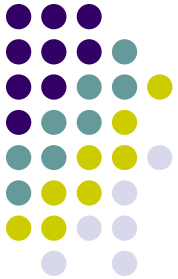
- 29-Apr-11
  - DateFormat df = DateFormat.getDateInstance(DateFormat.DEFAULT);
- 9:18:27 AM
  - DateFormat tf = DateFormat.getTimeInstance(DateFormat.DEFAULT)
- Wednesday, April 12, 2000 9:18:27 o’clock AM EDT
  - DateFormat dtf = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL)

String → Date



- DateFormat.parse( )
  - can throw ParseException, NullPointerException, StringIndexOutOfBoundsException
- Code examples
  - `Date d;`
  - `DateFormat df;`
  - `df = DateFormat.getDateTimeInstance(DateFormat.MEDIUM , DateFormat. MEDIUM);`
  - `d = df.parse("12-Apr-00 2:22:22 PM");`

String → Date



- `df = DateFormat.getDateTimeInstance(DateFormat.LONG, DateFormat. LONG);`
- `d = df.parse("April 12, 2000 2:22:22 PM EDT ");`

- `df = DateFormat.getDateTimeInstance(DateFormat.FULL, DateFormat.FULL);`
- `d = df.parse("Wednesday, April 12, 2000 2:22:22 o'clock PM EDT");`

java.text.SimpleDateFormat

extends DateFormat

- Locale aware
- Easier than DateFormat
- Constructors
  - SimpleDateFormat( )
  - SimpleDateFormat(“formattingPattern”)
    - “MM dd, yyyy”
  - SimpleDateFormat”(formattingPattern”, locale)
    - Locale.German, Locale.French, etc.
  - SimpleDateFormat(“formattingPattern”, specialSymbols)

SimpleDateFormat

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue

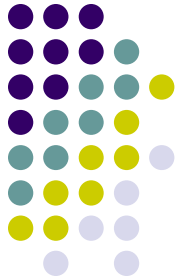
SimpleDateFormat

a	Am/pm marker	Text	`PM`
H	Hour in day (0-23)	Number	`0`
k	Hour in day (1-24)	Number	`24`
K	Hour in am/pm (0-11)	Number	`0`
h	Hour in am/pm (1-12)	Number	`12`
m	Minute in hour	Number	`30`
s	Second in minute	Number	`55`
S	Millisecond	Number	`978`
z	Time zone	General time zone	`Pacific Standard Time; PST; GMT-08:00`
Z	Time zone	RFC 822 time zone	`-0800`

SimpleDateFormat patterns

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, "yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o''clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700
"yyyy-MM-dd'T'HH:mm:ss.SSSZ"	2001-07-04T12:08:56.235-0700

Calendar



- Object broken into discrete fields.
  - Also represented by static fields in the Calendar class
- Using these fields, values may be obtained or set using the *get/set* methods.

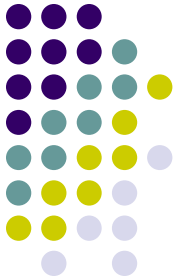
Calendar

- a default Calendar object
  - Calendar cal = Calendar.getInstance();
  - System.out.println("The current date and time is " + cal);
- conversion to a Date
  - Date date = cal.getTime();

Calendar - Gets and sets

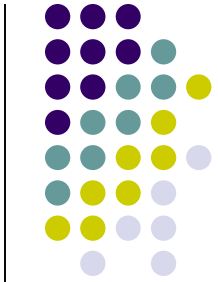
- int year = cal.get(Calendar.YEAR);
- cal.set(Calendar.YEAR, 2014);
- cal.set(Calendar.MONTH, Calendar.FEBRUARY);
- cal.set(Calendar.DAY_OF_MONTH, 2);
- cal.set(Calendar.HOUR, 13); // military style
- cal.set(Calendar.MINUTE, 8);
- cal.set(Calendar.SECOND, 23);

Calendar – date moving



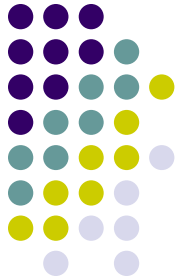
- add using September 3, you get:
- cal.add(Calendar.DATE, -3);
  // -3 days = August 31
- rolling back 3 days does not use other fields
- cal.roll(Calendar.DATE, -3);
  // -3 days = September 30

Calendar – leap year



- boolean leapYear =
  ((GregorianCalendar)cal).isLeapYear(2012);

Leap year



- boolean isLeapYear = (
  (year % 4 == 0) &&
  (year % 100 != 0) ||
  (year % 400 == 0));
- System.out.println(year + “ is
  (isLeapYear ? “” : “not “) + “ a leap year.”

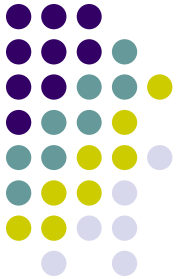
java.util.GregorianCalendar

- Converts a point in time to a calendar's month, day, and year
- Created for the current time zone.
- Create an object for the current system time:
  - GregorianCalendar now = new GregorianCalendar();

GregorianCalendar

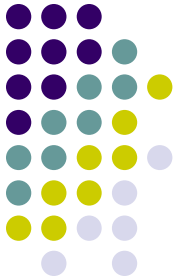
- Create an object for a specific date or time:
  - GregorianCalendar halloween = new GregorianCalendar(2011, Calendar.OCTOBER, 31);
  - GregorianCalendar nextCSI = new GregorianCalendar(2011, Calendar.OCTOBER, 31, 21, 0) // 2100 hours = 9 pm
- Locale aware

GregorianCalendar



- Setting the parts of the object:
  - GregorianCalendar johnsbirthday = new GregorianCalendar();
  - johnsbirthday.set(Calendar.YEAR, 1972);
  - johnsbirthday.set(Calendar.MONTH, Calendar.MAY);
  - johnsbirthday.set(Calendar.DATE, 20);
  - johnsbirthday.setTimeZone(TimeZone.getTimeZone(“MST”));
- other constants: ERA, DAY_OF_MONTH, DAY_OF_WEEK, WEEK_OF_MONTH, DAY_OF_WEEK_IN_MONTH, AM_PM, HOUR, HOUR_OF_DAY, MINUTE, SECOND, MILLISECOND

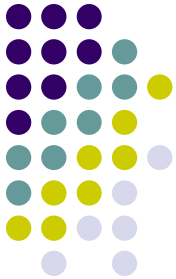
Exercise (LastDayOfMonth)



- DateFormat df
- = DateFormat.getDateInstance(DateFormat.FULL, Locale.GERMAN);
- GregorianCalendar dayIn2012 =
- new GregorianCalendar(2012, Calendar.JANUARY, 1);

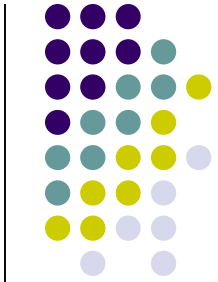
- for (int i = 0; i < 12; i++) {
- dayIn2012.roll(Calendar.DATE, -1);
- System.out.println(df.format(dayIn2012.getTime()));
- dayIn2012.add(Calendar.DATE, +1);
- }

java.sql



- java.sql.Date and java.sql.Time
  - equals( ) may not work due to missing ms
  - Oracle doesn’t store ms
- java.sql.Timestamp
  - more accurate with getNanos( )
  - long time = timestamp.getTime() + timestamp.getNanos() / 1000000;
  - a Date() may equals( ) a Timestamp but not vice versa

Joda-Time

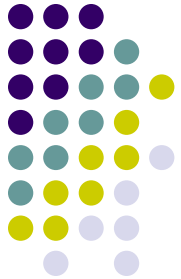


- http://www.joda.org/joda-time/
- Basic immutable classes
  - Instant - an instantaneous point on the time-line, timestamp of an event
  - DateTime - replacement for JDK Calendar
  - LocalDate - a local date without a time (no time-zone)
  - LocalTime - a time without a date (no time-zone)
  - LocalDateTime - a local date and time (no time-zone)

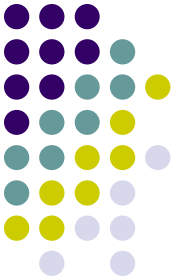
Joda-Time - DateTime

- DateTime dt = new DateTime(new Date( ))
- DateTime dt2 = dt.plusHours(2);
- dt.monthOfYear().getAsText()
- dt.monthOfYear().getAsShortText(Locale.FRENCH)
- dt.year().isLeap()

Joda-Time timespans

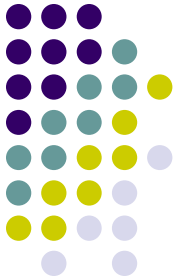


- Interval
  - holds a start and end date-time,
  - allows operations based around that range of time.
- Period – DST aware
  - 6 months, 3 days and 7 hours.
  - create a Period directly, derive it from an interval.
- Duration
  - Milliseconds
  - create a Duration directly, derive it from an interval.



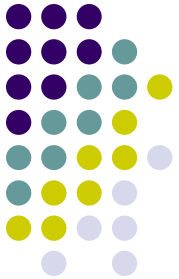
Joda-Time - formatting

- One method - forPattern(String)
  - Similar to SimpleDateFormat
- LocalDate date = LocalDate.*now();*
- DateTimeFormatter formatter
- = DateTimeFormatter.*ofPattern("d MMMM, yyyy");*
- String text = date.format(formatter);
- // "6 October, 2018"



java.time

- Based on Joda-time but many changes.

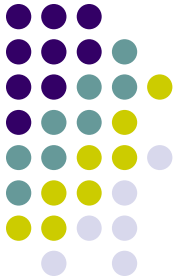


NUMBERS AND CURRENCY

Number objects

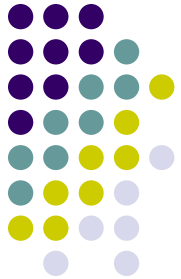
- Values are not objects but need a class for utility methods, a wrapper
- Byte, Short, Integer, Long, Float, Double
  - Character and Boolean are not subclasses
- Wrapper classes compose a value variable and provide
  - parse and value methods
  - max and min constants
  - constructor that uses parse on a String

Number objects



- int num = Integer.parseInt("42");
- System.out.println(num);
-
- Integer intObj= new Integer(382);

NumberFormat



- java.text
- Subclasses: ChoiceFormat, DecimalFormat
- myString = NumberFormat.**getInstance**().format(myNumber);
  - or break into two parts to reuse NumberFormat

NumberFormat

- factory methods
  - get**Currency**Instance( )
  - getInstance( )
  - get**Integer**Instance( )
  - get**Number**Instance
  - get**Percent**Instance( )
- factory methods overloaded
  - get<type>Instance(Locale)

NumberFormat

- set methods
  - setCurrency(Currency)
  - setGroupingUsed(boolean)
  - setMaximumFractionDigits(int) / setMinimumFractionDigits(int)
  - setMaximumIntegerDigits(int) / setMinimumIntegerDigits(int)
  - setRoundingMode(RoundingMode)

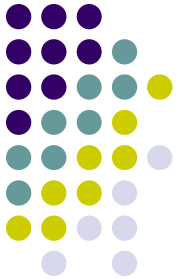
Exercise (NumberFormatTest)

- Built in formats
- NumberFormat cf = NumberFormat.getCurrencyInstance();
- NumberFormat pf = NumberFormat.getPercentInstance();
- NumberFormat nf = NumberFormat.getNumberInstance();

- System.out.println("The amount is " + cf.format(15.99));
- System.out.println("Your chance is " + pf.format(.25));
- System.out.println("The number is " + nf.format(2424));

Exercise (DecimalFormatTest)

- Custom formatting
- DecimalFormat df1 = new DecimalFormat("#,###.##");
- DecimalFormat df2 = new DecimalFormat("0,000.00");
- DecimalFormat df3 = new DecimalFormat(“$#,##0.00");
- System.out.println(df#.format(123456789.876654321));
- System.out.println(df#.format(1234));
- System.out.println(df#.format(0.1234));



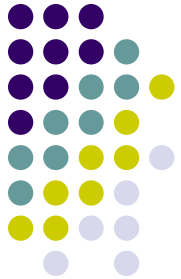
Floats/doubles not for $

- example showing precision limit
  - float a = 8250325.12f;
    float b = 4321456.31f;
    float c = a + b;
  - System.out.println(NumberFormat
    .getCurrencyInstance().format(c));
  - System.out.println("Float's max is " +
    NumberFormat.getIntegerInstance().format(Float.
    MAX_VALUE));

BigDecimal

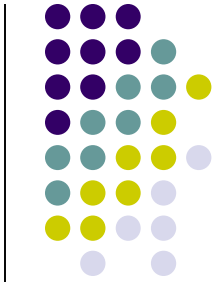
- uses an array to store each digit up to usable memory
- correct example
  - BigDecimal a1 = new BigDecimal("8250325.12");
    BigDecimal b1 = new BigDecimal("4321456.31");
    BigDecimal c1 = a1.add(b1);
  - System.out.println(NumberFormat .getCurrencyInstance().format(c1));

BigDecimal



- flexible rounding and scaling
- instance methods
  - abs( ), min( ), max( )
  - add( ), subtract( ), multiply( ), divide( )
  - intValue( ), longValue( )
  - floatValue( ), doubleValue( )
  - round(MathContext) – predefined IEEE standards of 32, 64 or 128 bit formats with HALF_EVEN rounding modes

BigDecimal



- class variables
  - // internal rounding
  - public static final MathContext ROUND_CURRENCY = new MathContext(6, RoundingMode.HALF_DOWN);
  - // display rounding
  - public static final NumberFormat CURRENCY_DISPLAY = NumberFormat.getCurrencyInstance(Locale.*GERMANY*);

BigDecimal

- BigDecimal amountOne = new BigDecimal("123.45");
- BigDecimal amountTwo = new BigDecimal("67.89");
- System.out.println(amountOne);
- System.out.println(amountTwo);
- BigDecimal result1 = amountOne.add(amountTwo);
- BigDecimal result2 = amountOne.subtract(amountTwo);
- System.out.println("Adding = " + result1);
- System.out.println("Subtracting = " + result2);

BigDecimal

- BigDecimal result3 = amountOne.multiply(amountTwo);
- BigDecimal result4 = amountOne.divide(amountTwo, ROUND_CURRENCY);
- System.out.println("Multiplying = " + result3);
- System.out.println("Dividing = " + result4);

- System.*out.println("---------- Currency displays");*
- System.out.println(CURRENCY_DISPLAY.format(result3));
- System.out.println(CURRENCY_DISPLAY.format(result4));

Regular expressions in Java

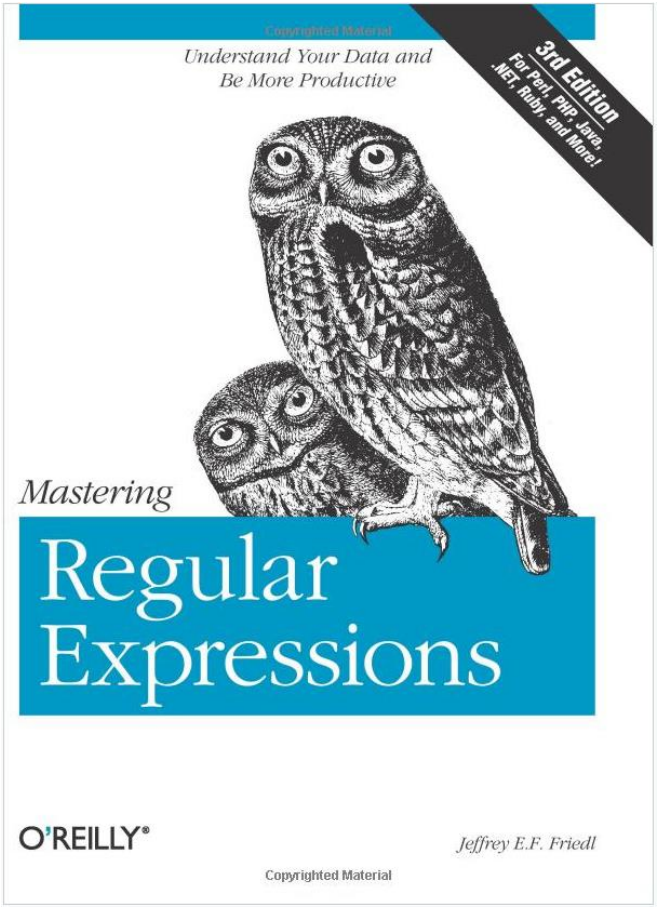
REGEX

Regular expressions (1.4)

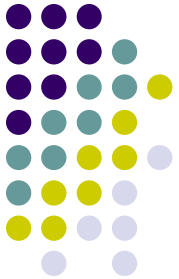
- A pattern matching language
  - Used in all popular languages as a DSL
- Regular expression
  - A pattern of characters that describes a set of strings.
- java.util.regex
  - follows the Perl flavor

Books

- *Mastering Regular Expressions*
  Jeffrey E. F. Friedl



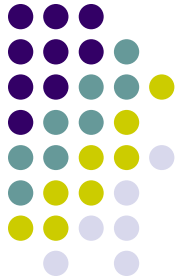
Web sites



- http://regexlib.com

- http://regexr.com
- http://gskinner.com/RegExr/
- http://regex101.com/

Metacharacters

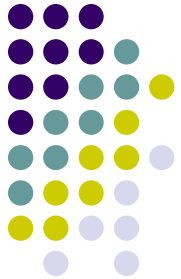


- Wildcards
  - period - any one character
- Position
  - ^ - beginning of a line / sequence
  - $ - end of line / sequence
    - Java matches beginning of sequence unless multi-line mode is turned on. Then it matches a \n.

Metacharacters

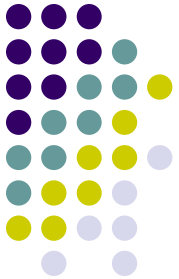
- [ ] - a subset / class of characters – c[lh]a
  - [^] - not this subset
- \ - an escape character
  - \. - the period character
  - \\ - the backslash character
  - In Java, you will have to escape the escape character in your strings! (\. ➔ \\.)
- ( ) - used to group choices
- | - OR e.g. (y | Y | ey | eY | Ey | EY)

Exercise



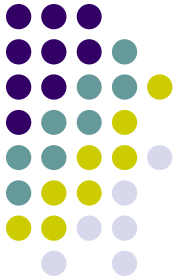
- **fun, food, phone, funny, typhoon, farm, fan, flood, Phineas, muffin**
- f.n
- ^f.n
- [fph][ph].n
- f[au]n
- ph[son].
- (f | ph) (o | oo)

Quantifiers (greedy)



- For cardinality / multiplicity
- Will consume the most characters as possible when making a match
- Follows the character / wildcard / subset
  - .?
  - .+
  - .*
  - .{5}

Quantifiers (greedy)



- ? - zero or one
- + - one or more
- * - zero or more
- {#} - exactly # times
- {#,} - at least # times
- {#,##} - between # and ## times

Characters

- x - The character x
- \\ - The backslash character
- \b - A word boundary
- \B - Not a word boundary

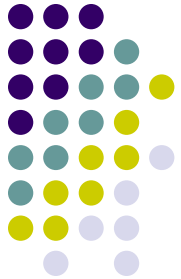
Characters

- \0n - The character with octal value 0n (0 <= n <= 7)
- \0nn - The character with octal value 0nn (0 <= n <= 7)
- \0mnn - The character with octal value 0mnn (0 <= m <= 3, 0 <= n <= 7)
- \xhh - The character with hexadecimal value 0xhh
- \uhhhh - The character with hexadecimal value 0xhhhh

Characters

- \t - The tab character ('\u0009')
- \n - The newline (line feed) character ('\u000A')
- \r - The carriage-return character ('\u000D')
- \f - The form-feed character ('\u000C')
- \a - The alert (bell) character ('\u0007')
- \e - The escape character ('\u001B')
- \cx - The control character corresponding to x

Character classes



- A grouping of characters that a character must belong to for a match
  - [abc] - a, b, or c (simple class)
  - [^abc] - Any character except a, b, or c (negation)
  - [a-zA-Z] - a through z or A through Z, inclusive (range)
  - [a-z-[bc]] - a through z, except for b and c: [ad-z] (subtraction)
  - [a-z-[m-p]] - a through z, except for m through p: [a-lq-z]{4}

Predefined Character Classes

- The word boundary character is not a predefined character class and can not be used like these.
  - . - Any character (may not match an EOL)
  - \d - A digit: [0-9]
  - \D - A non-digit: [^0-9]
  - \s - A whitespace character: [\t\n\x0B\f\r]
  - \S - A non-whitespace character: [^\s]
  - \w - A word character: [a-zA-Z_0-9]
  - \W - A non-word character: [^\w]

Lookaround

- Check but not consume
- When replacing text, this removes the character from what was not replaced.
  - (?=x) - see if x is the next character
  - (?=[xyz]) - see if x, y, or z is the next character
  - (?<=x) - see if x is the previous character
  - (?!x) - see if x was not the next character
  - (?<!x) - see if x was not the previous character

Capturing groups

- Results of a match are saved in one result
  - ((fun) (fon))
- Groups numbered by counting open parentheses from left to right.
- Group 0 is always the entire expression

Match flags - embedded / flag

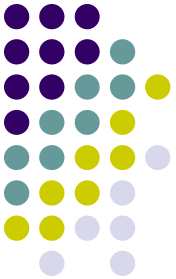
- (?i) CASE_INSENSITIVE
  - case-insensitive matching
  - (?i)ph.\w
- (?m) MULTILINE
  - the expressions ^ and $ match just after or just before, respectively, a line terminator or the end of the input sequence. By default, these expressions only match at the beginning and the end of the entire input sequence.
- (?im)expression - to combine

Match flags - embedded / flag

- (?s) DOTALL
  - the expression . matches any character, including a line terminator. By default this expression does not match line terminators. s is a mnemonic for "single-line" mode, which is what this is called in Perl.
- (?u) UNICODE_CASE
  - When enabled by the CASE_INSENSITIVE flag, case-insensitive matching is done in a manner consistent with the Unicode Standard. By default, case-insensitive matching assumes that only characters in the US-ASCII character set are being matched.

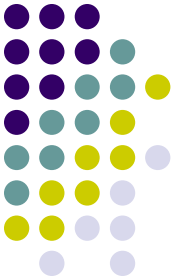
Match flags - flag only

- CANON_EQ
  - Enables canonical equivalence. When this flag is specified then two characters will be considered to match if, and only if, their full canonical decompositions match. The expression "a\u030A", for example, will match the string "å" when this flag is specified. By default, matching does not take canonical equivalence into account.



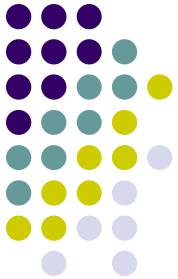
Exercise

- public static String text = "It was the best of times, it was the worst of times,\n"
- + "it was the age of wisdom, it was the age of foolishness,\n"
- + "it was the epoch of belief, it was the epoch of incredulity,\n"
- + "it was the season of Light, it was the season of Darkness,\n"
- + "it was the spring of hope, it was the winter of despair,\n"



Exercise (RegexEvaluator)

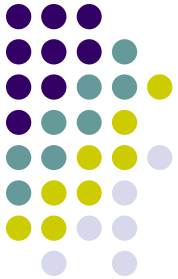
- + "we had everything before us, we had nothing before us,\n"
- + "we were all going direct to Heaven, we were all going direct the other way--\n"
- + "in short, the period was so far like the present period,\n"
- + "that some of its noisiest authorities insisted on its being received,\n"
- + "for good or for evil, in the superlative degree of comparison only.";



Exercise (RegexEvaluator)

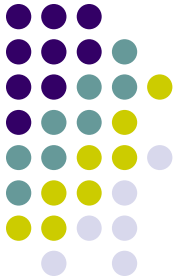
- import java.util.regex.*;

- public static void outputMatches(String _input) {
  - Pattern p = Pattern.compile(_input);
  - System.out.println("\nMatches using " + p.pattern() + "\n---------------");
  - Matcher m = p.matcher(text);
  - while (m.find()) { System.out.println(m.group()); }
- }



Exercise (RegexEvaluator)

- public static void main(String[] args) throws Exception {
- outputMatches("put pattern here");
- }

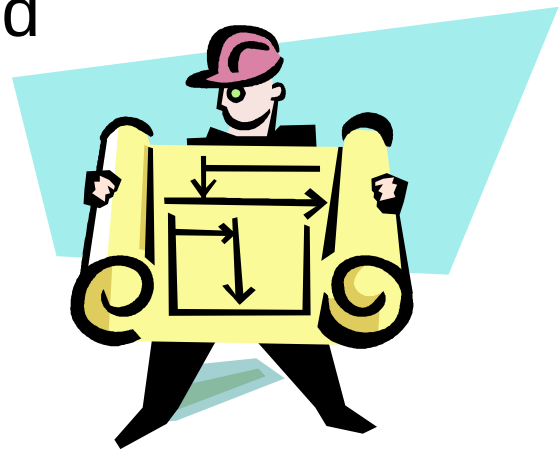


Exercise (RegexEvaluator)

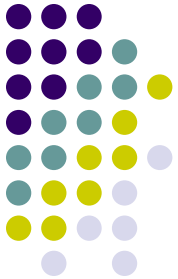
- Find these patterns:
  - od
  - .*od
  - o.*d
  - o.{5}d
  - (?i)o.{5}d
  - o.\w*
  - Same as the last one with a space before it
- Write patterns to find:
  - The line with the letters "it" at the beginning (not a capital I). (4 matches)
    - And try case insensitive too! (5 matches)
  - words ending in "s“ (20 matches)
  - words with a capital letter (4 matches)

Pattern class

- represents a regular expression that is specified in string form in a syntax similar to that used by Perl.
- compiles into a Pattern class object
  - can be used with multiple Matcher objects
  - original expression can be retrieved



Pattern class

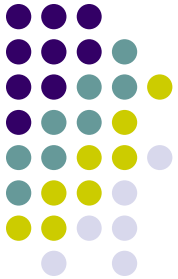


- Pattern p = Pattern.compile("cat");
- Pattern p = Pattern.compile("\\bcat\\b");
- Pattern p = Pattern.compile("(?mi)cat");
- Using a bit mask match flag
  - Pattern p = Pattern.compile("cat", Pattern.CASE_INSENSITIVE);
  - Pattern p = Pattern.compile("cat", Pattern.CASE_INSENSITIVE + Pattern. UNICODE_CASE);

Pattern - split( )

- Creates an array of strings based on match
  - Pattern.compile(**regex**).split(string)
  - Pattern.compile(**regex**).split(string, n)
  - “ a string”.split(**regex**)
- String inet = "127.0.0.1";
- String[] parts = inet.split("\\.");
- for (String part: parts) System.out.println(part);

Exercise (Splitter)



- Pattern p = Pattern.compile("[,\\s]+");
- String[ ] result =

  p.split("one,two, three four , five");
- for (int i=0; i<result.length; i++) {

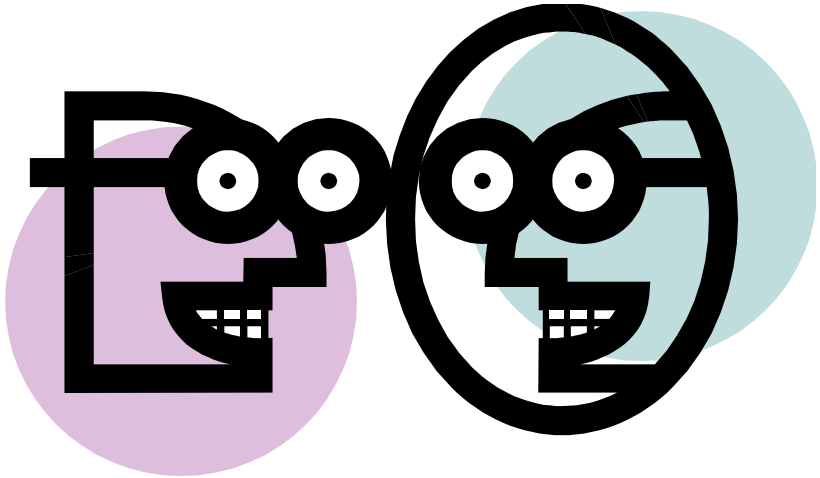
  System.out.println(result[i]);
- }

Pattern.matches( )

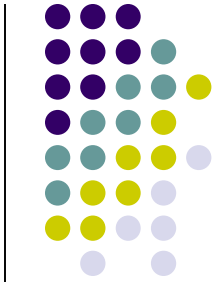
- when just used once
- matches(String regex, CharSequence input)
- boolean b = Pattern.matches("a*b", "aaaaab");
- Pattern.matches(zipRegex, “text to match”)
  - zip code regex = \d{5}(-\d{4})?
- Pattern.matches(HrefUrlRegex, “text to match”)
  - URL regex = (?<=href=")(.*?)(?=")

Matcher class

- For resusable patterns
- Pattern p = Pattern.compile (**regex**)
- Matcher m = p.matcher (**charSequence**)



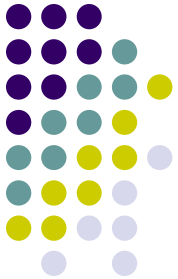
Matcher class



- Similar to Pattern match( ) but more reusable

- Pattern p = Pattern.compile("a*b");
- Matcher m = p.matcher("aaaaab");
- boolean b = m.matches();

Matcher class - find( )



- Pattern p = Pattern.compile("a.*b");
- Matcher m = p.matcher("ab aiiiiib aaaaaaaaaaaab");
-
- while (m.find()) {
- System.out.println(m.group());
- }

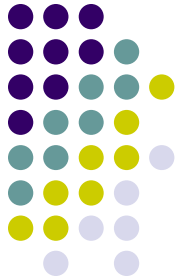
Matcher - Capturing groups

- String group( )
  - Returns the input subsequence matched by the previous match
- String group(int group)
  - Returns the input subsequence captured by the given group during the previous match operation.
- int groupCount()
  - Returns the number of capturing groups in this matcher's pattern.

Example (CharBufferExample)

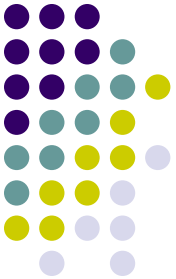
- Pattern p = Pattern.compile("//.*$", Pattern.MULTILINE);
- Matcher m = p.matcher(cb);
- while (m.find()) {
- System.out.println("Found comment: "+m.group());
- }

Matcher class



- String replaceAll(String replacement)
  - Replaces every subsequence of the input sequence that matches the pattern with the given replacement string.

Example (ControlCharacterRemover)



- Pattern p = Pattern.compile("{cntrl}");
- Matcher m = p.matcher(fileWithControlChars);

- String result = m.replaceAll("");

Matcher class

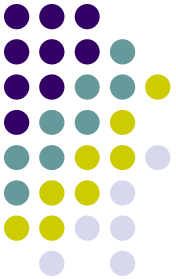
- Matcher appendReplacement(StringBuffer sb, String replacement)
  - Implements a non-terminal append-and-replace step i.e. appends everything up to the next match and the replacement for that match
- StringBuffer appendTail(StringBuffer sb)
  - Implements a terminal append-and-replace step i.e. appends the strings at the end, after the last match.

Match and append

- Pattern cat = Pattern.compile ("cat“)
- Matcher m = p.matcher(“one**cat**two**cat**three”);
- m.appendReplacement(string, “dog”)
- 1st appendReplacement( ) appends onedog.
- 2nd appendReplacement( ) appends twodog
- appendTail( ) appends three, resulting in the string “onedogtwodogthree”

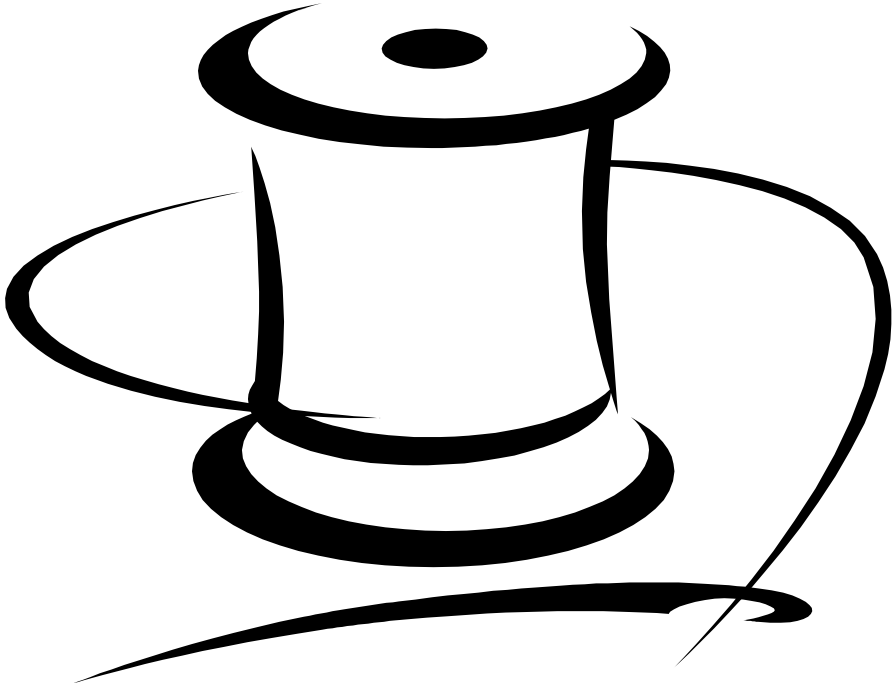
Exercise

- Write code to replace every occurrence of the word cat with dog in an input string in an example program.
- Modify the program so that **cat**, **Cat**, and **cats** are replaced only by **dog** and **dogs** just by modifying the regular expression.



Scanner 1.5

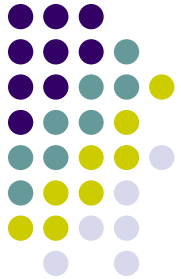
- parse primitive types & strings using regex
- use for console in
  - Scanner sc = new Scanner(System.in);
    int i = sc.nextInt();
- use with regex
  - String input = "1 fish 2 fish red fish blue fish";
    Scanner s = new
    Scanner(input).useDelimiter("\\s*fish\\s*");
    System.out.println(s.nextInt());
    System.out.println(s.nextInt());
    System.out.println(s.next());
    System.out.println(s.next()); s.close();



Executing multiple workflows concurrently

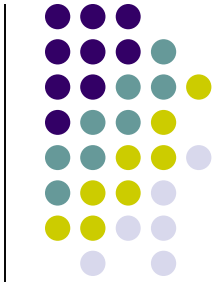
THREADS

Concepts



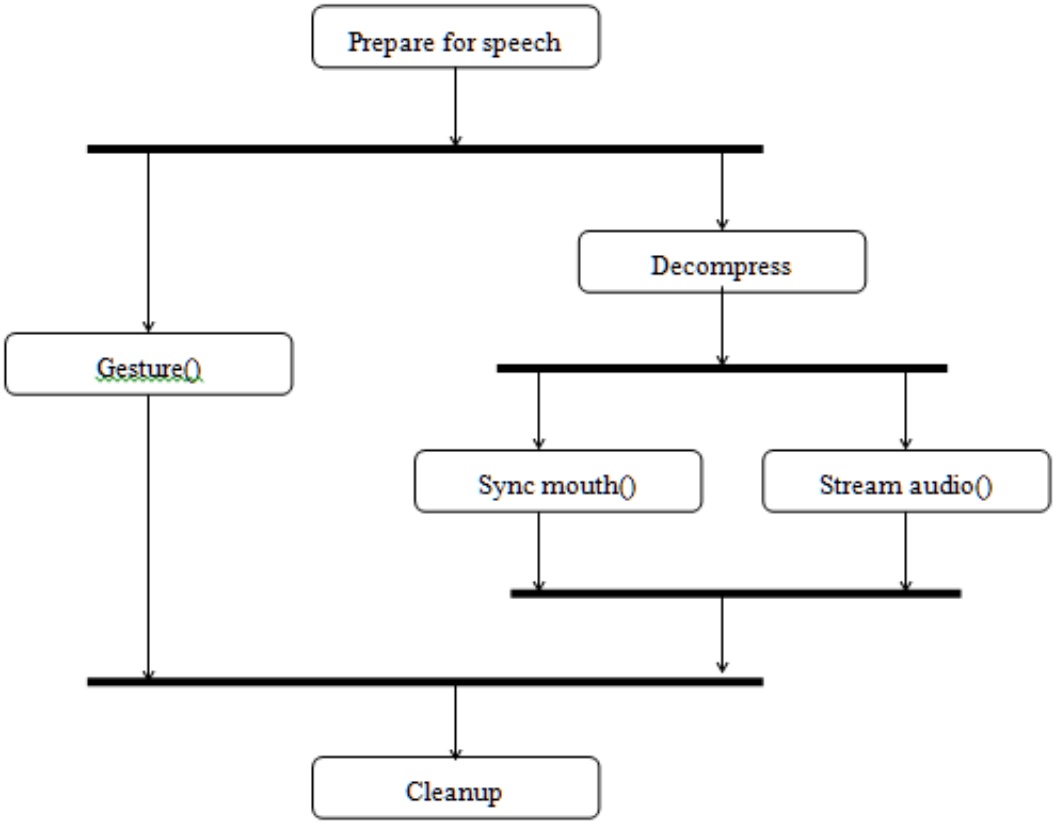
- Threads are independent workflows executing in the processor.
- Processor dependent
- Each thread uses resources but can also speed up total execution
- Threads are managed by the VM – not you through the code.
- Java has Object based thread support

Definitions 1

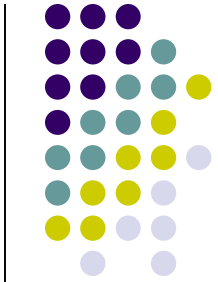


- **thread** - (from thread of control) a single path of execution through a program, dynamic model, or other representation of control flow.
- **fork** - the place in a thread of control where the process becomes a multiple threaded process
- **join** - the place in a thread of control where the process merges with another thread.

UML activity diagram

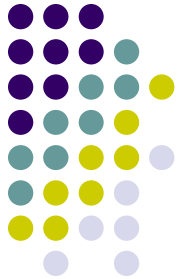


Thread basics



- You can’t pick when a thread starts or stops exactly.
- The first thread is the **main** thread.
  - Other threads fork from main.
- Threads work in the same process with
  - shared code
  - shared data

Thread class

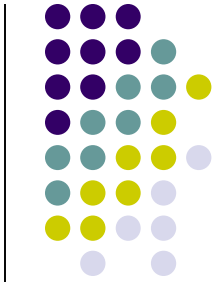


- A Thread object is useless by itself.
  - It has an empty run( ) method which is where the logic goes.
  - Thread t = new Thread();
- Threads can have names
  - Thread t = new Thread(“A Thread with a name”);

Threads with inheritance

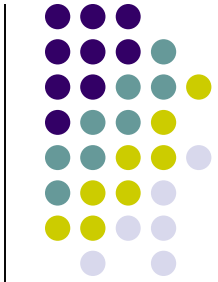
- Override the run( ) method with a subclass
  - class PrintNumbers **extends Thread** {
    - @Override
    - public void **run( )** {
    - //do something when started
    - }
  - }
- Create the active object
  - PrintNumbers pn = new PrintNumbers( );

Definitions 2



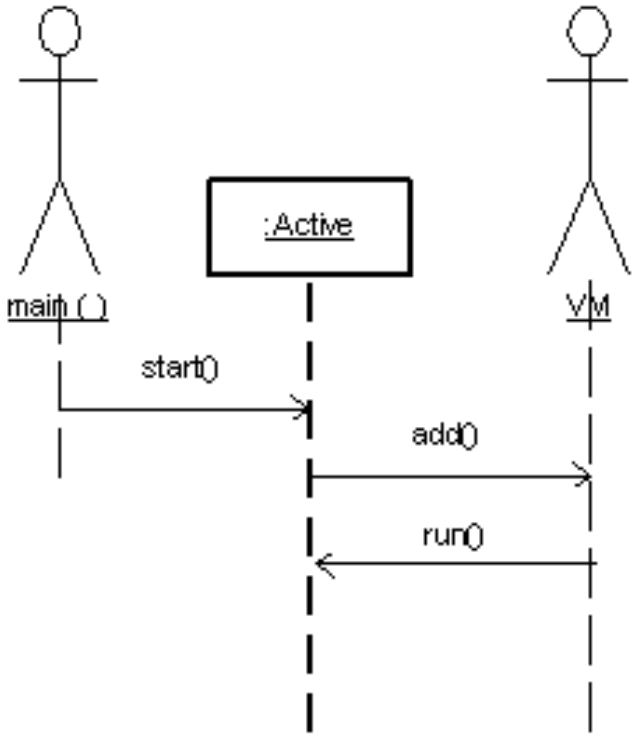
- **active class** - a class whose instances are active objects. In UML, the class box is drawn with a thicker line.
- **active object** - an object that owns a thread of control and can initiate control activity
- **passive object** - an object that does not have its own thread of control. Its operations execute under a control thread anchored in an active object.

Running threads

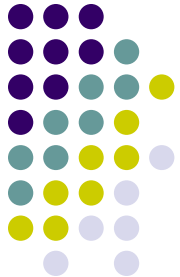


- You don’t. The VM calls the run( ) method.
- You place it in a runnable state by calling the **start**( ) method.
  - AThread t = new AThread( );
  - t.start( );
- When the VM chooses the thread to run, it calls the run( )

UML sequence diagram

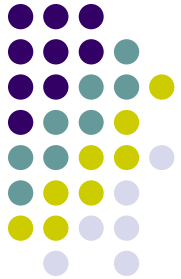


Four states



New	Runnable		Blocked			Dead
	Ready	Running	Sleeping	Waiting	Blocking I/O	
Before you call start( ).	VM will run a thread when it wants, after start( ) is called. A list of threads to be run is maintained.		Many java.io package methods let the thread run behind the scenes until its done while another thread is chosen to run by the VM. Methods sleep( ) and wait( ) require a try-catch block.			Has exited the run( ) method normally or because of an uncaught exception. Can not be restarted.

Thread properties



- boolean isAlive( ) – runnable, not blocked
- String getName( )
- void setName( )

Exercise

`control.threads.ThreadExample`

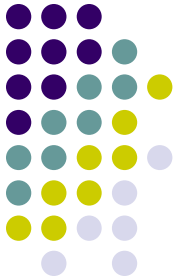
- public class ThreadExample extends Thread {
  - String whatToPrint;
  - public ThreadExample(String _whatToPrint) {
    - this.whatToPrint = _whatToPrint;
  - }
  - public void run() {
    - for (int j = 0; j < 100; j++) { // how many to print
      - for (long i = 0; i < 500000000; i++) { // 500,000,000
      - } // waste time to slow down
      - System.out.print(this.whatToPrint);
    - }
  - }
  - public static void main(String[] args) {
    - ThreadExample a = new ThreadExample("*** ");
    - ThreadExample b = new ThreadExample("111 ");
    - ThreadExample c = new ThreadExample("--- ");

    - a.start(); b.start(); c.start();
    - System.out.println("\nMain thread completed.");
  - } }

Threads by interface

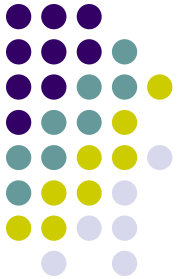
- Forced to use because of single inheritance
- Useful for thread reuse
- Uses the Runnable interface
  - one method – run( )
  - becomes the argument to a Thread constructor

The target



- public class RunnableClass extends AnotherClass implements Runnable {
  - @Override
  - public void run() {
    - // reusable code for any thread
  - }
- }

The target container

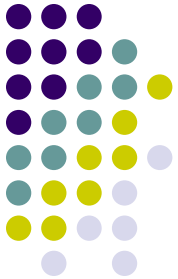


- RunnableClass runnableTarget = new RunnableClass();
- Thread t = new Thread(runnableTarget);
- t.start();

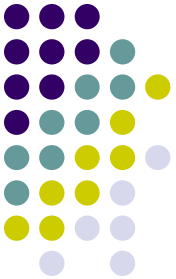
Common thread pattern

- public class RunnableClass extends AnotherClass implements Runnable {
  - Thread t;
  - public void run() {
    - // code
  - }
  - public void doTheThreadThing() {
    - this.t = new Thread(this);
    - this.t.start();
  - }
- }

Threads with anonymous class



- **Anonymous class** use
  - Thread t = new Thread() {
  - public void run( ) {
  - // code }
  - };



Threads with lambdas

- Thread t = new Thread(( ) -> process( ));

- Runnable r = ( ) -> process();
- Thread t2 = new Thread(r);

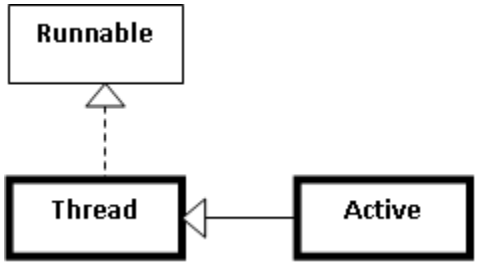
- t.start( ); t2.start( );

Enhancements

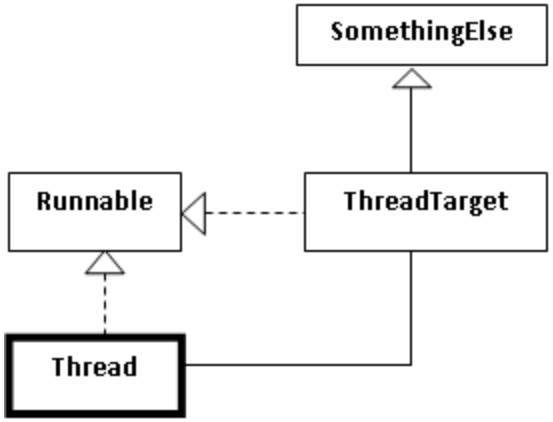
- <object>.**setPriority**( ) can alter the amount of time in each time slice by thread but is not reliable.
- Thread.**sleep**(long milleseconds)
  - pauses current thread
  - does not like to be woken up (interrupt( ))
    - throws InterruptedException
- Thread.**yield**( )
  - allows next thread to run
  - mostly for debugging

UML class diagrams

```
public class Active extends Thread {
    public run() {  }
}
```

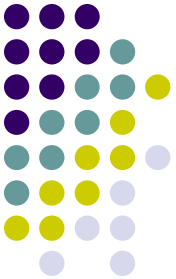



```
public class ThreadTarget extends
SomethingElse implements Runnable {
    Thread t = new Thread(this);
    public run() {  }
}
```




```
new Thread(aThreadTarget).start()
```

Exercise (SimpleFork)



- Write a class which forks two threads immediately. Have both threads print a iteration number up to 10. Delay each iteration by 1 second with
  - Thread.sleep(1000); // requires try/catch
- Options
  - create one object with two Thread variables
  - create two objects with one Thread variables

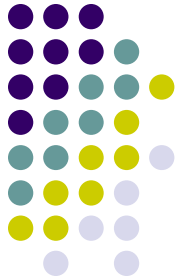
Timing threads

- System.currentTimeMillis( )
  - returns a long
- Profile your code
  - long time = System.currentTimeMillis();
  - // code to profile goes here
  - time = System.currentTimeMillis() - time;
  - System.out.println("Milliseconds taken: " + time);

Exercises

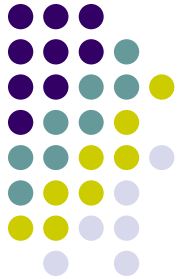
- Does the time you make a thread sleep equal the time that it really paused? Create a thread that does nothing but pause for 15 seconds and time it.
  - Do some other tasks on your computer while the program is running and see what effect that has.
- Use two threads to count down by twos so that the results are 10 8 6 4 2 9 7 5 3 1
  - now change to interleave: 10 9 8 7 6 5 4 3 2 1

Forks and joins



- **Forking** is creating a thread
- **Joining** is this thread waiting for the thread told to join to finish before continuing.

Exercise (JoinTest)



```
Thread t = new Thread() {
   public void run() {
     System.out.println("Reading...");
     try { System.in.read(); }
     catch (IOException e) { }
     System.out.println("Thread finished.");
     }
};
System.out.println("Forking.");
t.start();
System.out.println("Joining.");
try { t.join(); }
catch (InterruptedException e) { }
System.out.println("Main finished.");
```

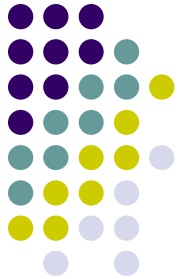
Exercise

- Create a loop of one billion doing nothing and time it. Then time two threads doing half a billion each. Then three threads doing a third of a billion each and so on.
- Check your processor name.

Code locks (monitors)

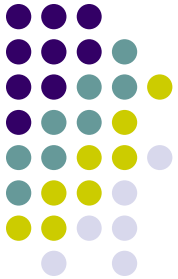
- Guarantee a code block to run unfragmented
  - synchronized (obj) {
  - // code that owns the locked object obj
  - }
- Guarantee a method to run unfragmented
  - synchronized methodName( ) {
  - //code that owns the object **this**
  - }

Synchronized blocks



- Take more time
  - Vector is, ArrayList is not synchronized
- Constructors can not be
- Static methods use the class object as a lock and not an invoking object if used
  - bad idea anyway
- **thread-safe** - a type of threaded application that controls concurrent access to its variables well.

Synchronized blocks

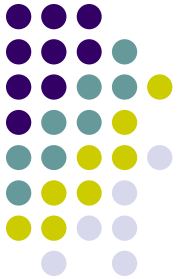


```
class LockExample {
    private Object lock1 = new Object();
    private Byte[] lock2 = new Byte[0];

    public void method1() {
           synchronized (lock1) {
              // code with access of one type
           }
    }
    public void method2() {
       synchronized (lock1) {
          /* code with same access as method1
          so methods will not run concurrently */
       }
    }
```

Synchronized blocks

```
    public void method3() {
        synchronized (lock2) {
          // code with access of another type so
          // methods 1 or 2 can run concurrently with
          // this one
        }
    }
}
```

Exercise

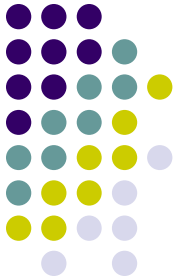


- **Demo**: NonSynchronizedBankAccount
  - Non-thread safe example
  - Thread safe example

Wait / notify

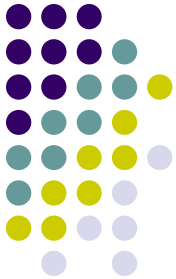
- used when waiting for a change
- lockObject.wait( ) – will block until given a notify( ) or notifyAll( ) is called
- lockObject.notify( )
  - will notify one object that is waiting and has been locked with this object
- lockObject.notifyAll( )
  - will notify all objects that are waiting and have been locked with this object
  - safer to use

Exercise



- No Wait / notify

Stopping / interrupting



- You don’t stop a thread.
  - stopping a thread is bad
  - stop( ), suspend( ), resume( ) are deprecated
- Interrupt it with aThread.interrupt()
- Interruptions set a flag checked with Thread.interrupted() - a little faster
  - or this.isInterrupted()
- An exception occurs if you interrupt a sleeping thread.

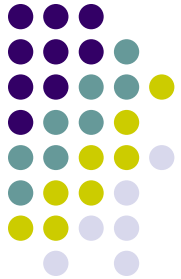
Thread scheduling 1.3

- Use Timer & TimerTask
- Timer methods
  - schedule(TimerTask, Date) – one time
  - schedule(TimerTask, Date, long) – recurring
  - schedule(TimerTask, long) – delay from now
  - schedule(TimerTask, long, long) – recurring after delay from now
  - scheduleAtFixedRate(TimerTask, Date, long)
  - scheduleAtFixedRate(TimerTask, long, long)

Thread scheduling

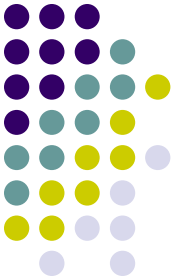
```
import java.util.*;
public class Reminder {
    Timer timer;
    public Reminder(int seconds) {
        timer = new Timer();
        timer.schedule(new RemindTask(), seconds*1000);
    }
    class RemindTask extends TimerTask {
        public void run() {
            System.out.println("Time's up!");
            timer.cancel(); //Terminate the timer thread
        }
    }
}
```

Thread scheduling



```
public static void main(String args[]) {
    System.out.println("About to schedule task.");
    new Reminder(5);
    System.out.println("Task scheduled.");
}
```

Thread scheduling by time



- `Calendar calendar = Calendar.getInstance();`
- `calendar.set(Calendar.HOUR_OF_DAY, 23);`
- `calendar.set(Calendar.MINUTE, 1);`
- `calendar.set(Calendar.SECOND, 0);`
- `Date time = calendar.getTime();`
-
- `timer = new Timer();`
- `timer.schedule(new RemindTask(), time);`

volatile

- Threads check other threads objects and values.
- If a thread updates a value and doesn't tell the other thread, you have a risk. These member variables can be marked **volatile.**
- Alan Holub doesn't recommend it because of unpredictability. Use explicit synchronization.

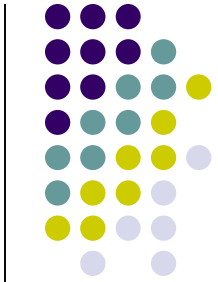
Thread groups

- Group threads by use with a ThreadGroup object
  - weak thread safety
  - obsolete
  - find another way by naming

Standard threads / priority

10	Reference Handler
8	Finalizer – so the VM can do finalize( )
6	AWT – EventQueue-0 – only for a **GUI**, also called the event thread and invokes event methods like actionPerformed( ), keyPressed( ), mouseClicked( ), and WindowClosing( ).
5	AWT - Windows – only for a **GUI**
5	main – the sequence of statements in the main( ) method.
5	Signal Dispatcher
5	SunToolKit.PostEventQueue-0 – only for a **GUI**
4	Screen Updater – only for a **GUI**

Use need-to-know



- Lock down the access as much as possible to your object data
  - private keeps other threads out
  - gets and sets can check for appropriate access
  - protect code blocks with **synchronized**
    - Any read with write dependence

Deadlock

- The toughest thread problem
- Data is locked by a thread before accessing another object. That object is locked by another thread wanting access to the first piece of data.
- Also called a race condition.
- Use state charts to find issues
- Use jCarder, Quest’s jProbe, etc.

Pipes

- To read or write data from other threads use
  - PipedReader
  - PipedWriter

Daemons

- Daemon threads are dependent on another thread.
- When the supporting thread dies, the daemons terminate.
- setDaemon(true)

Concurrency utilities - JDK 5,7

- the Fork / Join Framework
- java.util.concurrent
  - java.util.concurrent.atomic
  - java.util.concurrent.locks
- Features
  - better synchronization with semaphores and other patterns
  - better data exchange
  - better workflow management

Books

- ***Java Concurrency in Practice***
  - Brian Goetz, Tim Peierls, Joshua Bloch, Joseph Bowbeer, David Holmes, Doug Lea, Addison-Wesley Professional, 2006.
- ***Java Design***
  - *P*eter Coad, Mark Mayfield, and Jon Kern. Yourdon Press, 1999.
    - Chapter 4 – Design With Threads is an excellent discussion of four different designs for threaded applications.

Multithreading – theory vs practice

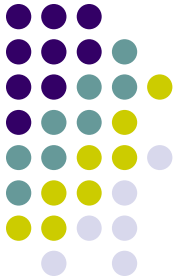
Controlling access to classes through objects

NESTED CLASSES

Inner classes

- Composition types – useful for coupling & scope
  - Static nested classes
    - Like a package name
  - Local classes
    - Declared in method code block
  - Basic nested classes
    - Declared in class code block
- Inheritance type – useful for convenience
  - Anonymous classes

Nested or Inner classes

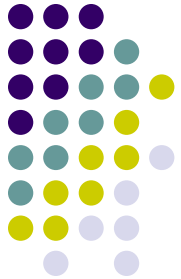


- Declaring a class inside of **another class**
  - class Hand { class Finger { } }
- Requires an object creation of the outer class first
  - Hand **right** = new Hand();
  - Finger rightThumb = **right**.new Finger();
- Used when both classes depend on each other and can’t live without each other

Creating an inner class object

- Inside the outer class
  - MyInner in = new MyInner( );
  - MyInner in = this.new MyInner( );
- Outside the outer class
  - MyOuter out = new MyOuter( );
  - MyOuter.MyInner in = out.new MyInner( );
    - or
  - MyOuter.MyInner in= new MyOuter( ).new MyInner( );

Accessing with inner classes

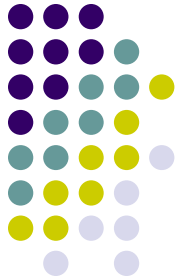


- The inner class object from inside the inner class
  - this
- The outer class object from inside the inner class
  - MyOuter.this

Static inner class

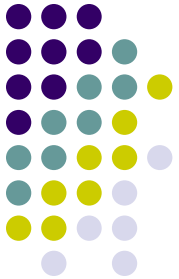
- Declaring a **static** class inside of **another class**
- Does **not** require an object creation of the outer class first
- Used when both classes depend on each other and can’t live without each other
- Outer class acts like a package name
  - Use a package instead

Static inner class syntax



- MyOuter.MyStaticInner in =
  new MyOuter.MyStaticInner( )
- Can not access fields of outer class

Local class

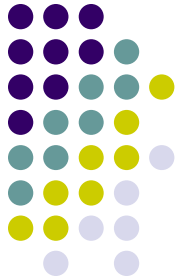


- Declaring a class inside of **a method**
- Objects created in a method have **local** scope
  - cease to exist after method completes.
- Used when the method uses the class objects exclusively for convenience

Anonymous class

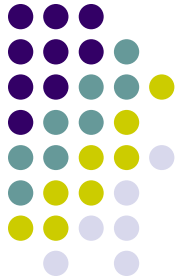
- Declaring an unnamed subclass when using a superclass constructor.
  - Either subclass or class implementing an interface
- For **one unique object** that would require a subclass
- A code block follows an object creation that contains the subclass code
  - Dog d = new Dog( ) {
    - public String getBark() { return “howooooooo”;}
  - } ;

Anonymous classes in Swing



- button1.addActionListener(new ActionListener( ) {
  - public void actionPerformed(ActionEvent e) {
    - aTextField.setText("Button 1 clicked");
  - }
- });

Exercise

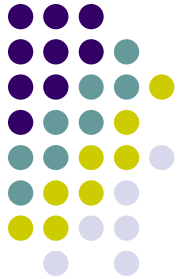


- Create a Dog class with a bark( ) method that just outputs “Bark.” Create a dog and make him bark.
- In the main method, create several anonymous classes to Dog to alter how your special dogs should bark.
  - Dog howler = new Dog() {void bark( ) {…}};

Created restricted datatype values

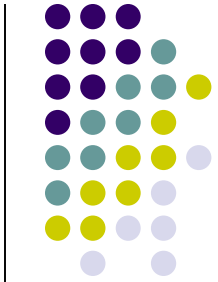
ENUMERATIONS JDK5

The enum



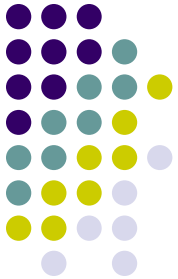
- Better than a set of named constants
- A class type with
  - constructors, methods, instance variables
- Features
  - type safety
  - objects, so collection aware
  - better understanding of value and meaning than constants
- Use when restricting types

Enum options



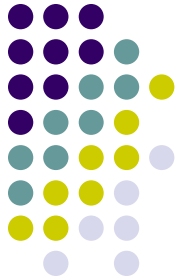
- EnumSet for sets of enums
- EnumMap for maps of enums
- Methods
  - clone( ), compareTo( ), equals( )
  - name( ), toString( )
  - ordinal( )
  - *EnumType.valueOf*(“toStringOutput”)
  - Arrays.*toString(EnumType.values( ))*

syntax



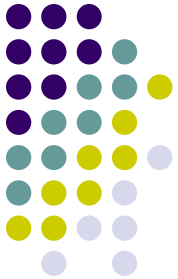
- declaration
  - enum Season {
    - WINTER, SPRING, SUMMER, FALL; }
- use
  - Season mySeason = Season.WINTER;
  - System.out.println(mySeason);
- output
  - WINTER

syntax



- A constructor is needed to create the declared enum values
  - required if you have instance variables
- Gets and sets are required for access to instance variables
- Enums inherit from the class Enum
- Use enums in switch statements

Example

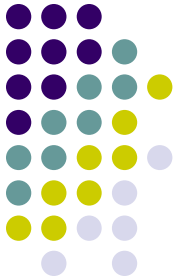


```
enum Coin {
    PENNY(1), NICKEL(5), DIME(10), QUARTER(25);
    private final int value;
    Coin(int value) {
        this.value = value;
    }
    public int getValue() {
        return value;
    }
}

System.out.println(aCoin.getValue());
```

Enums in the APIs

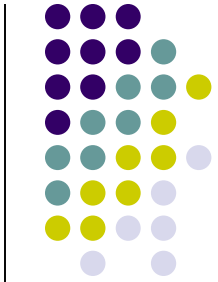
- Thread.State
- java.math.RoundingMode
- java.util.Formatter.BigDecimalLayoutForm
- java.net.Authenticator.RequestorType
- java.net.Proxy.Type

Command enums



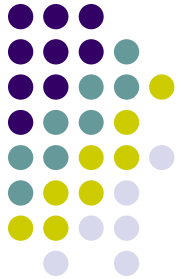
- public enum Toy {
  - DOLL( ) {
    - @Override public void execute( ) { System.out.println("I'm a doll."); } },
  - SOLDIER( ) {
    - @Override public void execute( ) { System.out.println("I'm a soldier."); } };
  - //template method
  - public abstract void execute();
- }

Command enums



- With the use of static imports, the client code calling this enumeration would look like:
  - SOLDIER.execute();
  - DOLL.execute();
- or better
  - getToy().execute();

Enums with reverse lookups



- Data is often "associated" with an enumeration.
- An enum is a class and the data then is a class field.
- Look up the enum value with a key.
  - Use a static Map

Enums with reverse lookups

- public enum Status {
  - WAITING(0), READY(1), SKIPPED(-1), COMPLETED(5);
  - private static final Map<Integer,Status> lookup = new HashMap<Integer,Status>( );
  - static { for (Status s : EnumSet.allOf(Status.class)) lookup.put(s.getCode(), s); }
  - private int code;
  - private Status(int code) { this.code = code; }
  - public int getCode( ) { return code; }
  - public static Status get(int code) { return lookup.get(code); }
- }

Exercises

- Write a class that represents the compass directions
- Write a class to represent the days of the week.
  - add a field to describe a task or origin of the day.
- Print out all the names and values.

200 End tasks

- Zip files and save/mail
- Certificate
- Evaluation