Substring question

s.substring(i, j) returns: ???

- first j characters of s starting at index i
- first j-1 characters of s starting at index i
- s starting at index i and ending at j
- s starting at index i and ending at j-1

object and classes and strings

- What's the difference between =, ==, and equals?
- How can I check whether a string s is the empty string?
- Is there a difference between the empty string and null?
- if the following legal?
  s.trim().toLowerCase().equals("saturday")

object and classes and strings

- data types
- primitive types vs. reference types.
- primitive types in Java: boolean, char, byte, short, int, long, float, and double
- arrays are object
- call by value

int compareTo(String anotherString)

let k be the smallest such index; then the string whose character at position k has the smaller value, as determined by using the < operator, lexicographically precedes the other string.

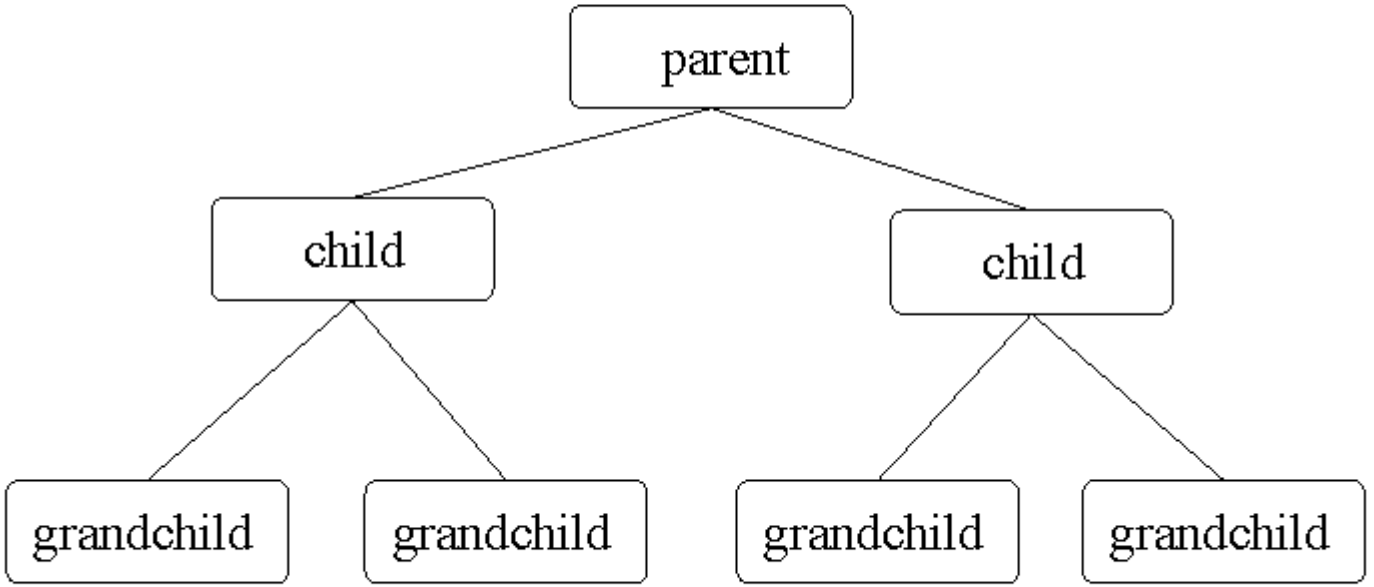
- this.charAt(k)-anotherString.charAt(k)

if there no such index

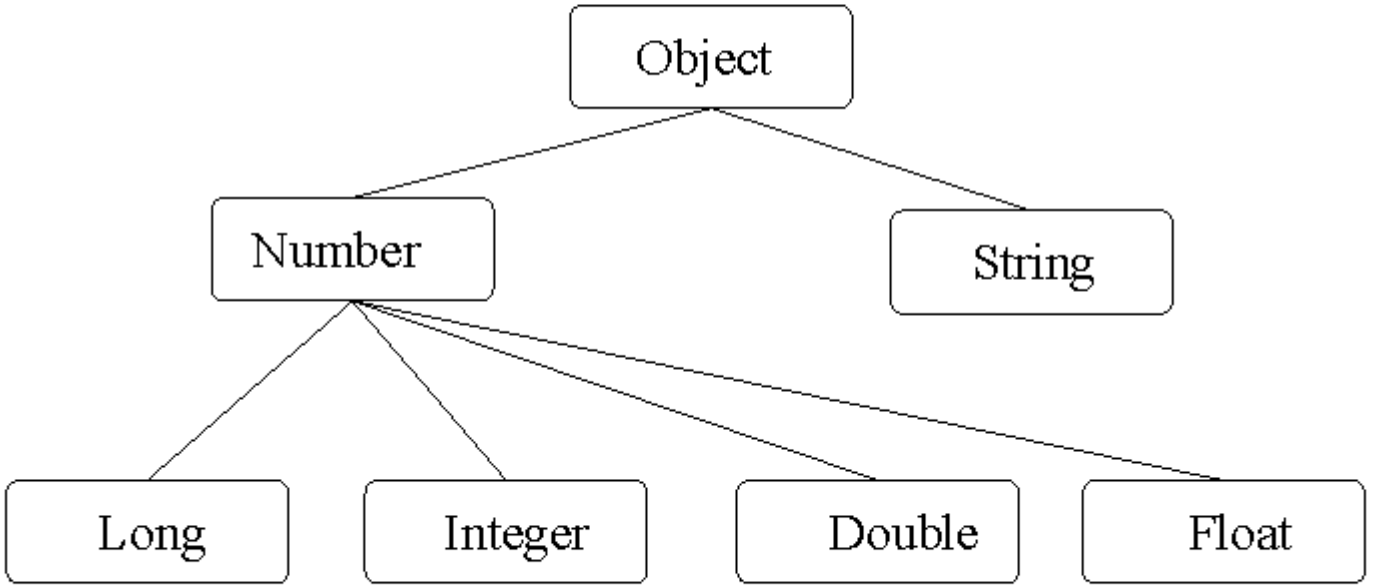
- this.length()-anotherString.length()

returns: *integer*

Chap 7 - Inheritance



An example from java.lang



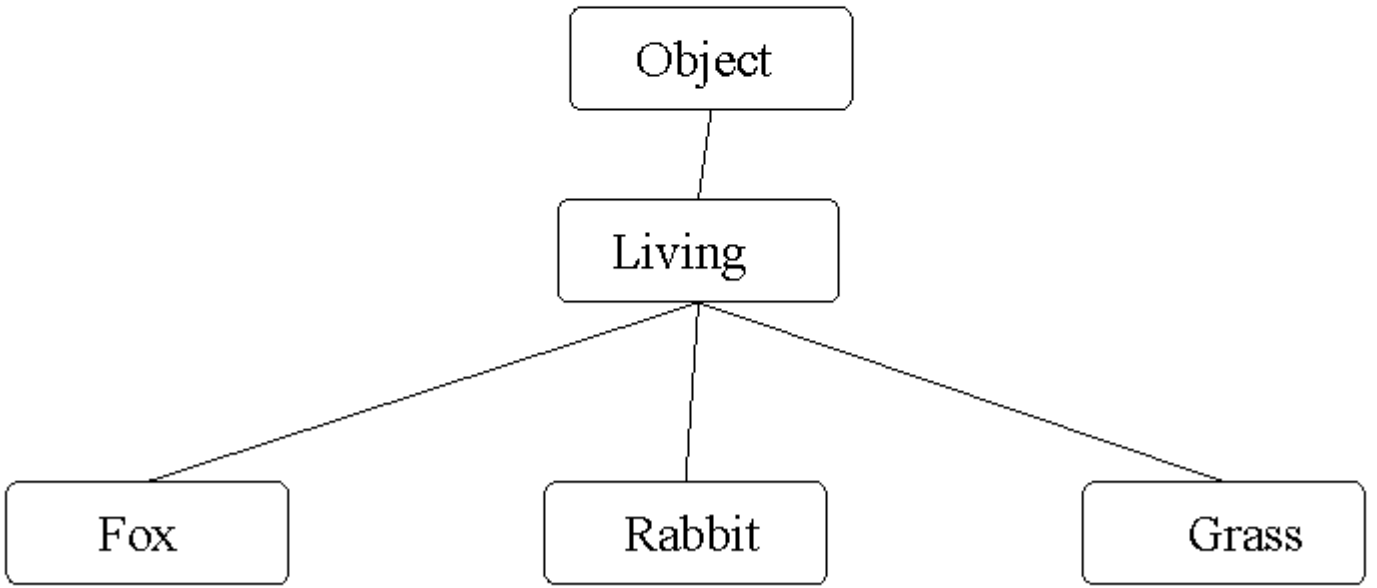
Dynamic Method Dispatch

```
//SuperClass.java - a sample super class
class SuperClass {
  public void print() {
    System.out.println( " inside SuperClass");
  }
}
//SubClass.java - a subclass of SuperClass
class SubClass extends SuperClass {
  public void print() {
    System.out.println( " inside SubClass");
  }
}
```

```
//TestInherit.java - overridden method selection.
class TestInherit {
  public static void main(String[] args) {
    SuperClass s =  new SuperClass();
    s.print();
    s = new SubClass();
    s.print();
  }
}
```

Predator-Prey: An abstract class

A simple simulation of an artificial ecology



```
//Living.java - the superclass for all life forms
abstract class Living {

  abstract Count getCount();
  abstract Living next(World world);
  abstract char toChar(); // character for this form
  void computeNeighbors(World world) {
    world.clearNeighborCounts();
    world.cells[row][column].getCount().set(-1);
    for (int i = -1; i <= 1; i++)
      for (int j = -1; j <= 1; j++)
        world.cells[row+i][column+j].getCount().inc();
  }
  int row, column; //location
}
```

```
class Fox extends Living {
  Fox(int r, int c, int a )
    { row = r; column = c; age = a; }
  Living next(World world) {
    computeNeighbors(world);
    if (Fox.neighborCount.get() > 5 ) //too many Foxes
      return new Empty(row, column);
    else if (age > LIFE_EXPECTANCY)   //Fox is too old
      return new Empty(row, column);
    else if (Rabbit.neighborCount.get() == 0)
      return new Empty(row, column);  // starved
    else
      return new Fox(row, column, age + 1);
  }

  public String toString(){ return "Fox age " + age; }
  char toChar() { return 'F'; }
  Count getCount() {  return neighborCount; }
  static Count neighborCount = new Count();

  private int age;
  private final int LIFE_EXPECTANCY = 5;
}
```

Why must getCount() and neighborCount be repeated in each subclass of Living? Why not just move these definitions to Living?

Answer: There is no way to write a method in Living, that accesses a static field in the subclasses of Living. We need a neighborCount for each of Fox, Rabbit, Grass, and Empty.

```
class Rabbit extends Living {
  Rabbit(int r, int c, int a )
    { row = r; column = c; age = a;}
  ...
  Count getCount() { return neighborCount; }

  static Count neighborCount = new Count();
  private int age;
  private final int LIFE_EXPECTANCY = 3;
}
```