Objects and Classes

• An **object** is a combination of **states** (i.e. data fields) and its **behaviors** (i.e. methods)

They are more complicated entities than primitive data types, can contain many properties of different types. It is a **user-defined** type.

• For example, suppose want to keep track of students at NYU. The student info should include: student id, classes taken, number of credits, entering year, etc.

• Without objects would need many years, of different types where info. for student $i$ in position $i$ in multiple arrays.

• With objects, we define a Student Class and put all these variables into the definition.

```java
public class Student{

    String name;       // name of student
    String id;         // N number for student
    int totalCredits;  // credits passed to date
    String [] classesTaken;  // e.g. CS101-007
    int enteringYear;   // year student started NYU
}
```

• Can make array of Student objects that holds all Students: **Student [] students;**

• Examples of behaviors might be **addingCourses** to **classesTaken**, **updateTotalCredits**, etc.
Objects and Classes

• An object is defined by a class. A class is a recipe (template) for how to make objects.
• A recipe can make many cookies. A class can make many students, each with different data.
• An object made this way is called an instance of the class, and its particular variables are called instance variables.

• Actually creating an object from the class is called instantiation. To do this, a special method is called in the class, called a constructor, which has the same name as the Class. A constructor specifies what a new object looks like and sets the initial data. Some possibilities are:

```java
Student s1 = new Student();  // create a student object, no arguments given
Student s1 = new Student("Alice", "N12345678", 32);  // create a student object, 3 arguments given
```

corresponding to the following two constructors (note use of overloading):

```java
// constructor with no arguments
Student() {
   name = "";  // default values for data
   id = "N00000000";
   totalCredits = 0;
}

// Constructor with 3 args passed
Student(String studentName, String studentID, int studentCredits) {
   name = studentName;
   id = studentID;
   totalCredits = studentCredits;
}
```
Objects and Classes

• A constructor is what is called when you execute `new`.
• A constructor is NOT a method. No return type, do not use void.
• Can have many constructors with different signatures (can be overloaded).
• A class usually provides a default no-arg constructor.
• The object is actually a reference variable to the place in ‘heap’ memory where data stored.
• A file may contain several classes. Only one can be `public`, and it must have same name as file.

How to access data from object?

• Suppose
  
  ```java
  s1 = new Student("Alice", "N27272727", 40);
  s2 = new Student("Bob", "N33333333", 38);
  ```

  Later need to retrieve data. Use dot operator `.` , which applies method to object before dot:

  ```java
  int credit1 = s1.getCredits();
  int credit2 = s2.getCredits();
  ```

  * `public int getCredits(){
    return totalCredits; // could also use this.totalCredits
  }`

  also possible to directly use `s1.totalCredits` if data is public but NOT good form.
Objects and Classes

```java
import java.util.Date;

public class Student{
    String name; // name of student
    String id; // N number for student
    int totalCredits; // credits passed to date
    String [] classesTaken; // e.g. CS101-007
    int enteringYear; // year student started NYU

    Student(){ // no args passed to constructor
        name = "";
        id = "";
    }

    // Constructor with 3 args passed
    Student(String studentName, String studentID, int studentCredits) {
        name   = studentName;
        id     = studentID;
        totalCredits = studentCredits;
    }

    public int getcredits(){
        return totalCredits;
    }

    public String getID(){
        return id;
    }

    public String getName(){
        return name;
    }
}
```

Student class so far ...
Objects and Classes

• If a class does not have a constructor, a default no-arg constructor with an empty body will be implicitly defined in the class. This is called a default constructor, and is automatically provided automatically only if no constructors are explicitly defined.

• There are default values provided for instance variables; NOT for local variables;

```java
public class TestStudent{
  public static void main(String[] args){
    Student s1 = new Student("Alice","N12345678",16);
    Student s2 = new Student();
    System.out.println("Student s1 = " + s1);
    System.out.println("Student s1 has id:" + s1.getName());
    System.out.println("Student s2 = " + s2);
    System.out.println("Student s2 has total credits: " + s2.getCredits());
  }
}
```

• If a reference variable has been declared, but does not reference any object you can assign it the special value NULL. For example, s1 = s2; // s2 object no longer accessible
  s2 = NULL; // make it clear

• Old s1 object called garbage; java reclaims heap space by performing garbage collection

• Another example: modify Student class to include Date object: date of enrollment.
Objects and Classes

- A class can have variables that are not associated with a specific instance of a class.
- These variables can be shared by all instances of the class. Called static variables or class variables.
- Static methods operate on static variables, never on instance variables.

Some examples:

- `Math.pow()` is a static class, there are no instance variables. No use of “new MathObject”, e.g.
- Suppose you have a `Circle` object and want its area. This refers to the area of a specific `Circle` (no such things as area in general). So getArea method must be called on the instance variable it refers to: `myCircle.getArea()`
- `public static void main` is a static method (invoked before any objects even created)
- add static variable to Student class:
  ```java
  static int totalNumStudents = 0; // initialize
  Student()
  {
    ...
    totalNumStudents++; // any instance can access in constructor and increment it
  } An instance has access to a static variables; not vice versa
  ```
Objects and Classes

Visibility modifiers:

• **public:** class, data, method visible to any class in any package (e.g. `main` method is public)

• **private:** data or methods can be accessed only within the class itself (e.g. `Math` class constructor is private so no objects can be instantiated).

• **default:** data and methods visible by any class in the same package

**Encapsulation:** use of private variables, for safety, protects against changes in implementation, and modification by users of software

How to access private data:

• **getters and setters:** methods provided to get and set private data also called accessor/mutator methods
  
  e.g.  
  ```
  double r = myCircle.getRadius();  
  or  
  myCircle.setRadius(4.2);  
  ```
Objects and Classes

- **this** reference: refers to the object itself (the object that invoked the method)

```java
public class TestCircle{
    Circle c = new Circle(5.4);
    double area = c.getArea();
}
```

- **toString** every class should have a method called `toString()` which displays the data in a class as a string. This allows it to be printed, e.g. for `Circle c` above, saying
  ```java
  System.out.println("My circle is" + c);
  ```
  automatically calls `toString` on `c` and displays it. Especially handy for debugging.

- **package** organizes classes by giving them same package name
  ```java
  package packageName;
  ```
  if no package statement, `default` package assumed (more on packages later)
Objects and Classes

- **equals**: are two objects equal? Suppose

  ```java
  public class Time{
    private int hour;
    private int minute;
    private int second;

    public Time(int hour, int minute, int second){
      this.hour = hour;
      this.minute = minute;
      this.second = second;
    }

    public boolean equals(Time t){
      return (this.hour == t.hour &&
              this.minute == t.minute &&
              this.second == t.second);
    }
  }
  ```

  ```java
  Time t1 = new Time(9,30,0);
  Time t2 = t1;
  Time t3 = new Time(9,30,0);
  ```

- Does t1 == t2?
- Does t1 == t3?
- **t1.equals(t3)** because their instance variables are equal. Common to write an **equals** method in a class to define the class definition of when 2 objects are equal. Default is `==`.