1. **Declared versus run-time type** State the declared type and the run-time (actual) type for each of the following:

   (a)
   ```
   PureBreed d = new Poodle();
   ```

   declared type _____ run-time type _____

   (b)
   ```
   Poodle p = new Poodle();
   ```

   declared type _____ run-time type _____

   (c)
   ```
   RuntimeException x = new IllegalArgumentException();
   ```

   declared type _____ run-time type _____

   (d)
   ```
   Number n = new Integer(5);
   ```

   declared type _____ run-time type _____

   (e)
   ```
   List<Integer> t = new ArrayList<Integer>();
   ```

   declared type _____ run-time type _____

2. **Casting**

   Assume that `Dog` has a non-abstract method named `bark` that all `Purebreed` dog classes override and `Mix` does not override. Suppose that a client has written the following method:

   ```
   public void speak(Dog d) {
     d.bark();
   }
   ```

   Which version of `bark` is run in each of the following:

   (a)
   ```
   PureBreed d = new Poodle();
   speak(d);
   ```

   (b)
   ```
   Poodle p = new Poodle();
   speak(p);
   ```

   (c)
   ```
   Dog d = new Poodle();
   speak(d);
   ```

   (d)
   ```
   Mix d = new Mix();
   speak(d);
   ```

   (e)
   ```
   Poodle d = new Poodle();
   speak((Dog) d);
   ```

3. **Abstract classes**

Consider the abstract `TurtleCommand` class from Lab 5:

```
public abstract class TurtleCommand {

        protected Turtle turtle;

        /**
         * Initializes the command with the Turtle instance that will
         * execute the command.
         *
         * @param turtle the Turtle instance that will execute the command
         */
        public TurtleCommand(Turtle turtle) {
                this.turtle = turtle;
        }

        /**
         * Causes a Turtle to execute the command.
         */
        public abstract void execute();
}
```

(a) Implement the `WalkCommand` class; recall that `Turtle` has a method `walk(double distance)`.

```
public class WalkCommand extends TurtleCommand {

        private double distance;

        /**
         * Initialize a walk command by a given distance
         * for the given turtle.
         *
         * @param turtle the Turtle to turn
         * @param distance the distance to walk
         */
        public WalkCommand(Turtle turtle, double distance) {



        }

        /**
         * Walk the turtle forward by the distance represented by
         * this command.
         */
        @Override
        public void execute() {


        }

}
```

(b) Implement the `PenColorCommand` class; recall that `Turtle` has a method `setPenColor(Color color)`.

```
public class PenColorCommand extends TurtleCommand {

        private Color color;

        /**
         * Initialize a pen color command by a given color
         * for the given turtle.
         *
         * @param turtle the Turtle to change the pen color on
         * @param color the new pen color
         */
        public PenColorCommand(Turtle turtle, Color color) {



        }

        /**
         * Change the pen color of the turtle to the pen color
         * represented by this command.
         */
        @Override
        public void execute() {


        }

}
```

4. **Invoking the parent version of a method**

   Consider a slightly different version of `TurtleCommand` where `execute` is not abstract and logs a string representation of the command (perhaps for debugging purposes):

```
public abstract class TurtleCommand {

        protected Turtle turtle;

        /**
         * Initializes the command with the Turtle instance that will
         * execute the command.
         *
         * @param turtle the Turtle instance that will execute the command
         */
        public TurtleCommand(Turtle turtle) {
                this.turtle = turtle;
        }

        /**
         * Causes a Turtle to execute the command and logs the
         * command.
         */
        public void execute() {
                // some code here that logs a string representation
                // of the command
        }
}
```

   Re-implement the `WalkCommand` class so that `execute` also logs the command; recall that `Turtle` has a method `walk(double distance)`.

```
public class WalkCommand extends TurtleCommand {

        private double distance;

        // assume constructor is already implemented

        /**
         * Walk the turtle forward by the distance represented by
         * this command and logs the command.
         */
        @Override
        public void execute() {



        }

        /**
         * Returns a string representation of this command.
         *
         * @return a string representation of this command
         */
        @Override
```

```
        public String toString() {
                return "walk " + this.distance;
        }

}
```